Model Coupling in Hydroinformatics Systems Through the Use of Autonomous Tensor Objects

Von der Fakultät für
Umweltwissenschaften und Verfahrenstechnik
der
Brandenburgischen Technischen Universität
Cottbus-Senftenberg
zur Erlangung des akademischen Grades eines
Doktor-Ingenieurs genehmigte Dissertation

vorgelegt von
Kunwar Vikramjeet Singh Notay, M.Sc.
aus
Neu Delhi, Indien

Gutachter: apl. Prof. Frank Molkenthin
Gutachter: Prof. Reinhard Hinkelmann
Tag der mündlichen Prüfung: 10. Juli 2015

Erklärung

Hiermit erkläre ich an Eides statt, dass ich diese Dissertation selbstständig verfasst und alle in Anspruch genomenen Hilfsmittel in der Dissertation angegeben habe. Ich erkläre auch, dass die Veröffentlichung dieser Dissertation bestehende Schutzrechte nicht verletzt.

Kunwar Vikramjeet Singh Notay
Cottbus, 24.03.2015

Abstract

There is an increased focus on interdisciplinary research in hydroinformatic related projects for applications such as integrated water resources management, climate change modelling, etc. The solution of common problems in interdisciplinary projects requires the integration of hydroinformatic models into hydroinformatic systems by coupling of models, enabling them to efficiently share and exchange information amongst themselves.

Coupling of models is a complex task and involves various challenges. Such challenges arise due to factors such as models required to be coupled together lacking coupling capabilities, different models having different internal data formats, lack of a coupling mechanism, etc. From the perspective of physics, different models may use different discretisations in space and time, operate on different scales in space and time, etc.

A model coupling concept using a coupling broker, that is independent from the coupled models, has been developed in this work and been implemented as a prototype for a software framework for coupling hydroinformatic models. It is based on the approach of tensor objects and the ideas of the OpenMI standard for model coupling. Tensor objects are a complete representation of physical state variables including dimensions, units, values, coordinate systems, geometry, topology and metadata. They are autonomous entities that can adapt themselves to the requirements of coupled models through operations such as scaling, mapping, interpolation in space and time, etc. The central entity in coupling is the Tensor Exchange Server, which acts as the coupling broker. It is responsible for defining the coupling mechanism, brokering the communication between the models and adapting the information to the requirements of the coupled models by taking advantage of the functionality provided by tensor objects. By fulfilling these roles in coupling, the coupling broker concept goes one step further than tools such as the OpenMI standard and facilitates the task of coupling models since each coupled model doesn't individually need to be adapted to be able to perform these tasks on its own.

The usefulness of the coupling broker concept for coupling models is demonstrated with the help of three application examples: firstly, a subsurface-flow model coupled with a model simulating metabolism in the hyporheic zone, secondly, a subsurface-flow model coupled with a surface-flow model and finally, an information

management system presenting the results of a hydrodynamic simulation of a section of the river Rhine. These examples demonstrate the extensibility and flexibility of the presented coupling concept, which can be used to couple multiple hydroinformatic models in hydroinformatic systems.

Kurzfassung

In Projekten der Hydroinformatik gewinnt der interdisziplinäre Aspekt zunehmend an Bedeutung wie z.B. bei Projekten zum IWRM (Integrated Water Resources Management) oder Modellierungen zum Einfluss des Klimawandels auf dem Wasserhaushalt. Zur Bearbeitung derartiger interdisziplinäre Problemstellungen ist eine Kopplung verschiedener Modelle der Hydroinformatik erforderlich, die u.a. eine gemeinsame Informationsverwaltung und einen gegenseitigen Informationsaustausch der Modelle realisiert.

Die Kopplung von Modellen ist eine komplexe Aufgabe, welche durch verschiedene Faktoren noch erschwert wird. Eine der größten Herausforderungen ist die oft mangelnden Kopplungsschnittstellen und Kopplungsfähigkeit von Modellen, welche sich anhand fehlender Kopplungsmechanismen und Unterschieden im internen Datenformat, in den verwendeten Raum- und Zeitskalen sowie der Diskretisierung von Raum und Zeit bemerkbar machen.

Im Rahmen dieser Dissertation wurde ein Modellkopplungskonzept, das sich auf ein modellunabhängiges Brokerkonzept basiert, für Modelle der Hydroinformatik entwickelt und mit einem Prototyp für ein Software-Framework umgesetzt. Dieser basiert auf einem objektorientierten Ansatz zur Modellierung physikalischer Zustandsvariablen und deren Beziehungen durch Tensorobjekte und den Modellkopplungsideen von OpenMI. Tensorobjekte sind eine vollständige Repräsentation physikalischer Zustandsgrößen, einschließlich Dimensionen, Einheiten, Größen, Koordinatensysteme, Geometrie, Topologie und Metadaten. Tensorobjekte sind autonome Entitäten, die sich an die Anforderungen der gekoppelten Modelle flexibel anpassen können. Ein Tensor Exchange Server übernimmt die zentrale Aufgabe der Kopplung der verschiedene Modelle als eine Art Kopplungsbroker. Zu seinen Funktionen zählen die Definition des Kopplungsmechanismus, die Kommunikation zwischen gekoppelten Modellen und die Anpassung dieser Informationen an die Anforderungen der Modelle (z.B. Zeit- und Ortsdiskretisierung) durch Ausnutzung der internen Anpassungsfähigkeit von Tensorobjekten. Diese Funktionsweise des Kopplungsbroker erübrigt eine individuelle interne Anpassung der einzelnen gekoppelten Modellen und geht somit einen Schritt weiter als konventionelle Kopplungsstandards wie OpenMI.

Die Vorzüge der Kopplung von Modellen durch das Kopplungsbrokerkonzept wird anhand drei Anwendungsbeispielen aufgezeigt: ein Untergrundströmungsmo-

dell gekoppelt mit einem Modell für Metabolismus in der hyporheischen Zone, ein Untergrundströmungsmodell gekoppelt mit einem Fließgewässerströmungsmodell und ein Informationsmanagementsystem mit 2D-Modellierungsergebnissen, GIS Modellen, Datenbankmodellen, etc. für einen Abschnitt des Flusses Rhein. Diese Beispiele zeigen die Flexibilität und Erweiterungsmöglichkeiten des hier vorgestellten Kopplungskonzepts, welches in der Kopplung verschiedenster Modelle in Hydroinformatiksystemen angewendet werden kann.

Acknowledgements

First and foremost I would like to thank my supervisors apl. Prof. Frank Molkenthin and Prof. Reinhard Hinkelmann for their continuous guidance and support, which made the successful completion of this thesis possible.

I would also like to express my gratitude towards my former colleagues Franz Simons, Leopold Stadler, Christian Adamczak, Ilhan Özgen, Clara Mendoza-Lera and Chi-Yu Li, for their help with various applications. Furthermore, I would like to thank all my friends and colleagues in both Cottbus and Berlin for the invaluable help they provided with academic, administrative and technical affairs during this thesis and for their company in informal settings. I would also like to thank every person who contributed, directly or indirectly, to this work.

I would also like to acknowledge the help provided by DFG for financially supporting this work in part.

Finally, I would like to thank my family for their patience and support for the entire duration of my work.

Contents

List of Figures

List of Tables

Listings

Nomenclature

Symbol	Description	Dimensions
α	Van Genuchten parameter α	$[\mathrm{M}^{-1}\mathrm{LT}^{2}]$
θ	Water content	$[-]$
μ	Dynamic viscosity of water	$[\mathrm{ML}^{-1}\mathrm{T}^{-1}]$
μ_α	Dynamic viscosity of phase α, where $\alpha \in \{w, n\}$	$[\mathrm{ML}^{-1}\mathrm{T}^{-1}]$
ν_t	Turbulent kinematic viscosity	$[\mathrm{L}^{2}\mathrm{T}^{-1}]$
ρ	Density of water	$[\mathrm{ML}^{-3}]$
ρ_o	Density of oxygen	$[\mathrm{ML}^{-3}]$
ρ_α	Density of phase α, where $\alpha \in \{w, n\}$	$[\mathrm{ML}^{-3}]$
ϕ	Porosity	$[-]$
ψ	Pressure head	$[\mathrm{L}]$
A	Cell area	$[\mathrm{L}^{2}]$
C	Chézy coefficient	$[\mathrm{L}^{-1/6}\mathrm{T}^{1}]$
C'	Bottom friction coefficient	$[\mathrm{L}^{-1}\mathrm{T}^{-1}]$
c	Concentration of dissolved oxygen	$[\mathrm{ML}^{-3}]$
$\mathbf{D}$	Dispersion tensor	$[\mathrm{L}^{2}\mathrm{T}^{-1}]$

$\mathbf{E}$	Component of the flux tensor along the x direction	$[-]$
$\mathbf{F}$	Flux tensor	$[-]$
$\mathbf{f}$	Flux vector through common edge between two adjacent cells	$[-]$
$\mathbf{G}$	Component of the flux tensor along the y direction	$[-]$
g	Gravitational acceleration	$[\mathrm{LT}^{-2}]$
$\mathbf{g}$	Gravitational acceleration vector	$[\mathrm{LT}^{-2}]$
h	Water depth	$[\mathrm{L}]$
i	Index of cell in the x direction	$[-]$
i	Time step index	$[-]$
j	Index of cell in the y direction	$[-]$
j	Bottom friction coefficient	$[-]$
K	Hydraulic conductivity	$[\mathrm{LT}^{-1}]$
$\mathbf{K}$	Permeability	$[\mathrm{L}^2]$
k	Index of edge within a cell	$[-]$
$k_{r\alpha}$	Relative permeability for the phase α, where $\alpha \in \{w, n\}$	$[-]$
l	Bottom friction coefficient	$[-]$
Δl_k	Length of k-th cell-edge	$[\mathrm{L}]$
m	Van Genuchten parameter m	$[-]$
n	Manning's roughness coefficient	$[\mathrm{L}^{-1/3}\mathrm{T}]$
n	Non-wetting phase	$[-]$
n	Time step index	$[-]$
n	Van Genuchten parameter n	$[-]$
$\hat{\mathbf{n}}_k$	Outward unit vector normal to k-th cell-edge	$[\mathrm{L}]$

p	Pressure	$[\mathrm{ML}^{-1}\mathrm{T}^{-2}]$
p_α	Pressure for the phase α, where $\alpha \in \{w, n\}$	$[\mathrm{ML}^{-1}\mathrm{T}^{-2}]$
p_{atm}	Atmospheric pressure	$[\mathrm{ML}^{-1}\mathrm{T}^{-2}]$
p_c	Capillary pressure	$[\mathrm{ML}^{-1}\mathrm{T}^{-2}]$
q	Source/sink/exchange flux term for water	$[\mathrm{ML}^{-3}\mathrm{T}^{-1}]$
q_α	Source/sink term for the phase α, where $\alpha \in \{w, n\}$	$[\mathrm{ML}^{-3}\mathrm{T}^{-1}]$
q_{avail}	Theoretical exchange flux if the whole quantitiy of water water above the surface of the current cell were to infiltrate	$[\mathrm{LT}^{-1}]$
q_o	Source/sink term for oxygen	$[\mathrm{M}^2\mathrm{L}^{-6}\mathrm{T}^{-1}]$
q_{poss}	Possible flux across the coupled interface under current simulation conditions	$[\mathrm{LT}^{-1}]$
q_w	Source/sink term for water	$[\mathrm{ML}^{-3}\mathrm{T}^{-1}]$
$\mathbf{S}$	Vector of source terms	$[-]$
S_α	Saturation for the phase α, where $\alpha \in \{w, n\}$	$[-]$
$S_{\alpha r}$	Residual saturation for the phase α, where $\alpha \in \{w, n\}$	$[-]$
S_e	Effective saturation of the wetting phase	$[-]$
Δt	Time-step size	$[\mathrm{T}]$
Δt_d	Current time-step size for DuMu$^{\mathrm{X}}$	$[\mathrm{T}]$
Δt_H	Current time-step size for HMS	$[\mathrm{T}]$
Δt_m	Time-step size for hyporheic zone metabolism model	$[\mathrm{T}]$
t	Time	$[\mathrm{T}]$
t_d	Time for current time-step for DuMu$^{\mathrm{X}}$	$[\mathrm{T}]$
t_{dlast}	Newest time for which the Tensor Exchange Server has values available for DuMu$^{\mathrm{X}}$	$[\mathrm{T}]$
t_{end}	Simulation end time	$[\mathrm{T}]$

t_H	Time for current time-step for HMS	[T]
t_{Hlast}	Newest time for which Tensor Exchange Server has values available for HMS	[T]
t_m	Time for current time-step for hyporheic zone metabolism model	[T]
$\mathbf{U}$	Vector of conserved quantities	$[-]$
u	Velocity component in the x direction	$[\mathrm{LT}^{-1}]$
v	Velocity component in the y direction	$[\mathrm{LT}^{-1}]$
$\mathbf{v}$	Flow velocity vector	$[\mathrm{LT}^{-1}]$
$\mathbf{v}_\alpha$	Darcy velocity for the phase α, where $\alpha \in \{w, n\}$	$[\mathrm{LT}^{-1}]$
w	Wetting phase	$[-]$
Δx	Length of cell in the x direction	[L]
x	x coordinate axis	$[-]$
x_d	Coordinates of coupled node in DuMu$^{\mathrm{X}}$	[L]
x_H	Coordinates of coupled node in HMS	[L]
x_m	Coordinates of coupled node in hyporheic zone metabolism model	[L]
Δy	Length of cell in the y direction	[L]
y	y coordinate axis	$[-]$
z	Geodetic head	[L]
z_B	Bottom elevation above datum	[L]

Chapter 1

Introduction

This chapter provides a foundation for the work presented in this thesis. It introduces the concepts and reasons for coupling models in hydroinformatic systems, the challenges involved in coupling models and so on. It begins by explaining the motivation behind the current work followed by a discussion on the problems involved in the coupling of hydroinformatic models. After these problems have been presented, the objectives of the current work are set out. The subsequent section defines some important terms, as used in this work. Finally, an outline of the subsequent chapters is provided.

1.1 Motivation

Hydroinformatics is a interdisciplinary domain incorporating a number of disciplines such as mathematics, physics, hydrology, hydraulics, hydraulic engineering, civil engineering, environmental engineering, information and communications technology (ICT), biology, chemistry, geophysics, telemetry and remote sensing among others. Due to this reason, projects related to hydroinformatics are often interdisciplinary in nature and involve experts from more than one of the disciplines mentioned above working in cooperation with each other.

The EU Water Framework Directive aims at achieving a “good status” of all waters, surface and subsurface, within a given time frame and to maintain this status in the future as well (European Union, 2000). Similarly, the EU Floods Directive aims to reduce and manage the risks that floods pose to human health, the environment, cultural heritage and economic activity through flood risk manage-

ment, which involves prevention of damage caused by floods to infrastructure, implementation of flood protection measures, preparing the population against floods, etc. (European Union, 2007a,b). The achievement of aims set down by these and similar legal frameworks requires an integrated river basin management approach involving simultaneous modelling of physical processes describing the complete hydrological cycle over the scale of the entire river basin. This has led to a further increase in the focus on interdisciplinary research in the field of hydroinformatics in recent times.

Working on interdisciplinary projects is a challenging task because of various differences in the backgrounds of the people involved, the models that they use, etc. Kragt et al. (2013) categorise such challenges into the following categories:

- **technology integration**, i.e. standardisation of data and models
- **knowledge integration**, i.e. integrating the know-how from different domains, different, work styles, epistemology, etc., and
- **team integration**, i.e. harmonising interactions of people with different cultures, belief systems, languages, etc.

To expand upon the first point "technology integration" put forward by Kragt et al. (2013), a number of different models are employed in any interdisciplinary project. These include, but are not limited to:

- **measurement models** for collecting information in the field, recording results of laboratory models, etc.
- **laboratory models** for physically modelling, e.g. flow of water around hydraulic structures, scale modelling of entire river basins, determination of physical parameters such as permeability, etc.
- **simulation models** for solving various hydroinformatic problems by process modelling with the aid of computers, based mainly on numerical methods but also using statistical, artificial intelligence, etc.
- **visualisation models** for visualisation of information in the form of diagrams, plots, maps, etc. either statically or dynamically with the aid of animations, virtual reality, user interaction, etc.

Within interdisciplinary hydroinformatics related projects, these models need to be integrated into hydroinformatic systems by coupling them together so that

they are able to share and exchange information with each other and to communicate with each other for control and operational purposes. This is necessary in order to be able to use them efficiently for the solution of common problems in an integrated manner. Integration of models into hydroinformatics systems is a complex task which must not only ensure the storage of information but also to make it available to each of the coupled models in a format which can be easily understood and effectively utilised by it. According to Knapen et al. (2013), integration of models involves the following:

- **semantic integration**: a common language and shared understanding between all models
- **methodological integration**: aligning different scientific methodologies and identifying required model improvements necessary for meaningful linkage so that data produced by one model are a meaningful input to another model irrespective of time and space scales
- **technical integration**: automating information interchange between models, making them jointly executable without human intervention

Information management, therefore, plays a very important role in the integration of models into hydroinformatic systems. It must fulfil all of the above-mentioned goals while ensuring the storage of information and its availability to each model in the system in a format that can be easily understood and effectively utilised.

1.2 Problem identification

The DFG Research Group "Natural Slope" (Hinkelmann and Zehe, 2006, 2013) was an interdisciplinary research project dealing with a creeping landslide along the mountain slope in Heumös near Ebnit, Austria. The project consisted of eight subprojects working on hydrology, subsurface hydraulics, continuum mechanics, geophysics and hydroinformatics with the aim of trying to better understand and explain the physical phenomena involved in slow-creeping landslides. One of the proposed methods for achieving this goal was the coupling of hydroinformatic models. However, as already mentioned in the previous section, this isn't such an easy task and similar to the experiences of other engineers, as described in the previous section, the following challenges were identified when trying to couple different models in the project:

- **Lack of coupling capabilities**: Most hydroinformatic models are developed without the aim of using them in a coupled set-up with other models. As a result, they are usually closed systems lacking capabilities for interacting with other models.
- **Different data formats**: There is no single unified standard for the representation of information with different models using different internal data structures for this purpose. As a result, models usually cannot directly understand or utilise information received from other coupled models and need to adapt this information into a format conforming with its internal data structure in order to be able to use it.
- **Different physical representation**: Models use not only different physical representation of physical state variables, e.g. hydraulic head vs. pressure, bu also different spatial and temporal discretisations, e.g. structured vs. unstructured meshes or explicit vs. implicit schemes.
- **Lack of coupling mechanism**: Although it is possible to couple models using various mechanisms such as file input/output or inter-process communication (Wikipedia, 2015b), there is a lack of a generalised model coupling mechanism making it is possible to couple models across different platforms, optionally running on different machines.
- **Different scales**: Different models work with information on different space and time scales, e.g. the width of macropores is typically in the centimetres range while the slow-creeping landslide takes place over an area of a few square kilometres. Similarly, infiltration processes are active over a time scale of seconds to hours while the slow-creeping landslide takes place over a period of years.
- **Need for physically meaningful coupling**: In loose coupling it is quite easy to create set-ups where the models are able to exchange information with each other but such exchange isn't always meaningful from the point of view of physics. As a result, checks and controls are required to catch certain obvious mistakes that can possibly be made during the coupling process, e.g. exchanging information with improper spatial or temporal discretisation, or worse, exchanging completely different physical state variables because semantic information about the exchanged information is missing or not taken into consideration.

- **Need for defining coupling interfaces**: Coupling of models involves identifying interfaces for the sharing or exchange of information among the coupled models and also the type of information that should be shared or exchanged. This requires an understanding of the physical processes involved in the coupling process in order to decide the nature of the information to be exchanged or shared among the coupled models, and the mechanism of such exchange.

1.3 Solution approach

Attempts have been made in the past to address these issues in the coupling of hydroinformatic models, e.g. the development of standards based on markup languages for the representation of hydrological information, e.g. SensorML (OGC, 2015b) and WaterML (OGC, 2015c) (see section 2.2.2), or for defining standardised interfaces for data exchange among coupled models, e.g. the OpenMI standard (Gregersen et al., 2007) (see section 2.3.2). However, there is still a lack of a single unified coupling mechanism that is able to address the above-mentioned problems. The current work attempts to develop one such coupling mechanism that is based on the use of tensor objects, as proposed by Molkenthin (2000), Molkenthin et al. (2009a,b) and inspired by the OpenMI standard. Tensor objects can be used for effective information management strategies for hydroinformatic systems in particular and for interdisciplinary engineering projects in general. Tensor objects are autonomous entities that contain a detailed representation of information and can exist independent from the models themselves. They can adapt themselves through operations such as scaling, mapping, interpolation, differentiation, integration, etc. This makes them ideal for use in information management and for coupling of models within hydroinformatics systems, since by taking care of the transformations required in sharing and exchange of information during the coupling of models, they help in reducing the overhead necessary for the individual models in order to couple them with other hydroinformatic models.

1.4 Objectives

The current work is based on the concepts of tensor objects, which provide a generalised representation of information, and on the OpenMI standard that describes strategies for the coupling of models. It aims to build upon these concepts to describe

a prototype for a software framework for model coupling within hydroinformatic systems with the following objectives:

- **Simplicity**: The framework should be based on concepts already familiar to people working with hydroinformatic models. The overhead required to use the framework for coupling existing models should also be kept at a minimum possible level.

- **Completeness**: It should be possible to fully represent the information from a physical state variable such as units, coordinate systems, geometry, topology, etc.

- **Autonomy**: The representation of information used by the coupling framework should be independent from the internal data structures of the coupled models and capable of independent existence.

- **Mobility**: It should be possible to freely exchange or share information among the coupled hydroinformatic models independent from whether those models run on the same machine or are coupled over a networked connection.

- **Adaptability**: Depending on the requirements of the coupled models, it should be possible to adapt information through operations such as scaling, mapping, interpolation, differentiation, integration, etc.

- **Fexibility**: The framework should make it possible to couple a variety of hydroinformatic models including simulation models, measurement models, laboratory models, statistical models, artificial intelligence models, etc.

- **Feasibility**: It should reduce the overhead required to make existing models ready for coupling by itself handling the adaptations required to adapt the shared or exchanged information according to the requirements of these models.

- **Modularity**: It should be possible to remove, replace, add new models to the coupling set-up in order to be able to adapt it to the modelling requirements.

- **Platform-independence**: It should be capable of coupling models running under different operating systems.

1.5 Definition of important terms

The following is a list of some important terms that are significant in the context of the current work along with their meanings, as they are used in the subsequent sections:

- **Coupling**: Coupling of models is the process of combining multiple models together in a set-up where the output from one model can be used as a meaningful input for another model, and vice versa, e.g. coupled surface and subsurface flow models that exchange information in the form of boundary conditions and/or sink/source terms.
- **Data**: Data is a collection of related items such as numbers, facts, etc. having a specific syntax and format, e.g. a series of measurements of rainfall data at a particular location.
- **Hyporheic zone**: The hyporheic zone is a spatially fluctuating ecotone between the surface stream and the deep groundwater (Boulton et al., 1998).
- **Hortonian runoff**: Hortonian runoff, often referred to as excess infiltration, occurs when the rainfall rate exceeds the saturated hydraulic conductivity of the land surface (Kollet and Maxwell, 2006).
- **Infiltration**: The process of movement of water from the earth's surface into the subsurface.
- **Information**: Information is a collection of data, its semantics and additional metadata, e.g. a rainfall time series including rainfall measurements, units of measurements, location of the instrument, etc.
- **Interface**: An interface is the means for accessing information stored in an object. According to Gamma et al. (1994), an object's interface characterizes the complete set of requests that can be sent to the object.
- **Metadata**: Metadata means data about data, i.e. a brief summarisation that provides a brief overview of the data (see section 2.1.1).
- **Metabolism**: The term metabolism, as used in the current work, refers to community metabolism. Community metabolism refers to the biological transfer of carbon and involves measurement of the basic ecological processes of production (via photosynthesis) and respiration (Gordon et al., 2004).

- **Model**: A model is an abstract and simplified representation of reality that serves some useful purpose in engineering, e.g. computing the flow of water in a river with the aid of computers by approximating the solution of partial differential equations. Models generally produce or process information during their operation.

- **Object**: An object is a representation of a real world item or a concept in the computer's memory. An object consists of two parts: information or state needed to describe the real world item or concept and methods or behaviour for working with that information like accessing or manipulating it.

- **Remote Procedure Call (RPC)**: A mechanism that allows a software process to invoke a procedure, subroutine or method which is not within its address space, i.e. one which is running in another process on the same machine or some other computer connected through a network.

- **Subsurface flow**: The flow of water below the earth's surface.

- **Surface flow**: The flow of water along the earth's surface.

- **Tensor**: Tensors are geometric objects that describe linear relations between vectors, scalars, and other tensors (Wikipedia, 2015m). According to Weisstein (2003b), tensors are generalizations of scalars (that have no indices), vectors (that have exactly one index), and matrices (that have exactly two indices) to an arbitrary number of indices.

- **Tensor objects**: Just like tensors are a generalised representation of numerical values, tensor objects are an autonomous and generalised representation of information containing scalars, vectors, tensors, etc. (see section 3.3). In this work tensor objects are used to represent physical entities within hydroinformatics models and systems.

- **Web services**: A web service is a piece of business logic, located somewhere on the internet, that is accessible through standard based internet protocols such as [HyperText Transfer Protocol] HTTP or [Simple Mail Transfer Protocol] SMTP (Chappell and Jewell, 2002).

1.6 Outline

The rest of this thesis is organised as follows:

The current state of the art in various technologies such as object-oriented programming, network communication services, modelling toolboxes and model coupling is discussed in chapter 2.

Chapter 3 describes the concepts behind tensor objects and coupling of models through such tensor objects.

In chapter 4 a software prototype that implements tensor objects and is capable of utilising them for model coupling is described.

Chapter 5 is devoted to practical examples demonstrating the versatility of tensor objects and their usefulness in model coupling.

Finally, a summary of the work is presented in chapter 6 along with an outlook for future research.

Chapter 2

State of the art

Hydroinformatics covers a wide variety of models. A number of different technologies may be employed for the coupling of the models. In this chapter, we take a look firstly at the current state of a subset of hydroinformatic models relevant to this work and secondly at the technologies relevant to model coupling in hydroinformatics.

2.1 Hydroinformatic models

Computer hardware and software have seen rapid development over the past several years. The rise of the "computing revolution" has also had an impact on the engineering community. The availability of increasingly powerful and smaller computing devices at ever lower prices has led to the rise of newer engineering disciplines such as hydroinformatics, that make extensive use of computing technology for solving hydro-engineering problems. The term "hydroinformatics" was originally coined by Mike Abbot, who defined it as 'the integration of computational hydraulics and of artificial intelligence' (Quevauviller, 2010). By elaborating slightly on this definition, hydroinformatics can be described as the branch of engineering that deals with the application of information and communications technologies in water-related engineering domains such as hydraulics, etc. Engineers working on hydroinformatics related projects often work with a large variety of models, such as simulation models and laboratory models for their work. The following sections describe some of these model types in detail.

2.1.1 Metadata models

The website of the Dublin Core Metadata Initiative (DCMI) states that '[t]he word "metadata" means "data about data". Metadata articulates a context for objects of interest – "resources" such as MP3 files, library books, or satellite images – in the form of "resource descriptions". As a tradition, resource description dates back to the earliest archives and library catalogs' (DCMI, 2015b). Like in the case of library catalogues, the "resource descriptions" or the metadata model provides a brief overview of the "resource" or the data. They help in searching for the data without having to go through the entire contents of the data itself. Many different metadata models for describing different kinds of data exist. Three of these standards: Dublin Core for generic "resources", ISO 19115 for geospatial data, and INSPIRE for geospatial data in Europe are discussed in the next sections.

Dublin Core

The following statement can be found in the metadata about the DCMI: '[t]he Dublin Core Metadata Initiative is an open forum engaged in metadata innovation and support the development of interoperable online metadata standards that support a broad range of purposes and business models' (DCMI, 2015a). Two metadata standards that have been developed and are maintained by the DCIM are:

- **Dublin Core Metadata Element Set**, which is 'a vocabulary of fifteen properties for use in resource description' (DCMI, 2012a). These properties are: title, creator, subject, description, publisher, contributor, date, type, format, identifier, source, language, relation, coverage, and rights (DCMI, 2012a) of the resource.
- **DCMI Metadata Terms** is the full set of all the metadata vocabularies and technical specifications maintained by the DCMI. These terms also contain the smaller set of terms that form the Dublin Core Metadata Element Set (DCMI, 2012b).

ISO 19115

ISO 19115 is the ISO standard for metadata related to geographic information. According to the ISO website, ISO 19115 'provides information about the identification, the extent, the quality, the spatial and temporal schema, spatial reference, and dis-

tribution of digital geographic data' (ISO, 2003). Furthermore, the website states that the standard is applicable to:

- 'the cataloguing of datasets, clearinghouse activities, and the full description of datasets
- 'geographic datasets, dataset series, and individual geographic features and feature properties'

Like in the case of the Dublin Core standard, the ISO 19115 standard defines a set of vocabulary for metadata, which in the case of ISO 19115 relates to geographic information.

INSPIRE

INSPIRE stands for INfrastructure for SPatial InfoRmation in the European community (EUR-Lex, 2007). It is an EU directive for 'establishing an infrastructure for spatial information in Europe to support community environmental policies, and policies or activities which may have an impact on the environment' (European Union, 2007d). INSPIRE is based on the following principles (European Union, 2007c):

- 'data should be collected only once and kept where it can be maintained most effectively
- 'it should be possible to combine seamless spatial information from different sources across Europe and share it with many users and applications
- 'it should be possible for information collected at one level/scale to be shared with all levels/scales; detailed for thorough investigations, general for strategic purposes
- 'geographic information needed for good governance at all levels should be readily and transparently available
- 'easy to find what geographic information is available, how it can be used to meet a particular need, and under which conditions it can be acquired and used'

For satisfying these principles the INSPIRE directive sets down requirements for metadata (EUR-Lex, 2008), network services, data-sharing, interporability, etc.

for spatial information that the EU member states need to implement (EUR-Lex, 2007).

2.1.2 Information models

An information model can be described as a structured collection of information. It may be simple in its composition, e.g. a collection of tables, spreadsheets, XML documents, etc. or may have a more complex composition, e.g. a Database Management System (DBMS) (see section 2.2.3). Some of the features of an information model include:

- metadata models (see section 2.1.1) providing a description of the information and helping in discovering information
- functionality for adding, removing and modifying information
- overview of the information, e.g. maximumn, minimum and average values, etc.
- functionality for transforming the information to be able to use it with different hydroinformatic models
- visualisation of information as graphs, tables, maps, etc.
- import or export of information from and to different formats
- web services (see section 2.2.4) for accessing information over the internet

Some examples of information models in hydroinformatics are Turtle (Molkenthin et al., 2009b), an information management system for interdisciplinary hydroinformatics related projects, NOKIS (KFKI, 2015), an information model for the North and Baltic seas, CUAHSI HIS (CUAHSI-HIS Project Team, 2015) for the sharing of hydrological data, WISKI (KISTERS, 2015) for acquisition, processing and archiving of information in hydroinformatics and Geoportal (Geoportal, 2015), a portal to geographic data based on the INSPIRE directive (European Union, 2007d).

2.1.3 Measurement models

Measurement models include instruments that are used to make measurements in the field, in laboratories, etc. for observation and monitoring. Examples of measurement models include sensors for measuring precipitation, wind, river cross-sections,

flow velocities, nutrients dissolved in water, etc. Measurement models may be built with functionality that enables them to send the measurement data over networks such as the internet, telecommunication networks, etc. Application examples are online monitoring systems using sensor networks maybe with real time control. Among other uses, measurement models provide information that is used to set up laboratory and simulation models, set their initial and boundary conditions, etc.

2.1.4 Laboratory models

Laboratory models are a scale-representation of a part of the reality existing in nature. They are used for simulating a natural physical phenomenon on a smaller scale in the laboratory. Examples of laboratory models in hydroinformatics include models simulating flow of water around hydraulic structures such as weirs, dams, etc. scale-representations of river catchments, etc. Since laboratory modelling is not the primary focus of this work, laboratory models are not described in further details here. However, they are mentioned here, since it is also possible to couple them with other hydroinformatic models, e.g. to set up the initial or boundary conditions from measurement models, to validate results of simulation models, etc.

2.1.5 Mathematical models

According to Ferziger and Perić (2002), '[t]he starting point of any numerical method [describing a physical phenomenon such as fluid flow] is the mathematical model, i.e. the set of partial differential or integro-differential equations and boundary conditions'. Therefore, mathematical models are quite important in the field of hydroinformatics as they serve as the foundation of a large number of hydroinformatic models. Examples of mathematical models include:

- Laplace and Poisson equations
- Navier-Stokes equations (Ferziger and Perić, 2002, Versteeg and Malalasekera, 2007)
- advection-dispersion equations
- Lattice Boltzmann methods (Succi et al., 2010)
- simplified models that are derived from the Navier-Stokes equations and are valid under different assumptions, e.g. Euler flow, potential flow, Boussinesq approximation, etc. (Ferziger and Perić, 2002)

- turbulence models, e.g. the Reynolds-Averaged Navier-Stokes (RANS) equation models, e.g. $k - \varepsilon$ model, Large Eddy Simulation (LES) models, e.g. Smagorinsky model, Direct Numerical Simulation (DNS) models, etc. (Ferziger and Perić, 2002, Versteeg and Malalasekera, 2007)

Coupling of hydroinformatic models is also possible through the coupling of mathematical models, as demonstrated by Gunduz and Aral (2005), Spanoudaki et al. (2009), Zhu et al. (2012), etc. However, the current work focusses on coupling of models through the use of software techniques and as a result, coupled mathematical models aren't described in any more details here.

2.1.6 Simulation models

A large number of simulation models exist in the field of hydroinformatics. Examples include models for simulating open channel and overland flow, subsurface flow models, water quality models, etc. These models use a variety of different solution techniques for solving the particular problem that they were designed to simulate. These include differences in (Ferziger and Perić, 2002, Österr. Wasser-und Abfallwirtschaftsverband, 2007, Versteeg and Malalasekera, 2007):

- the concept forming the basis of the simulation model, e.g. physically-based models, conceptual models, distributed or lumped models, black box models, artificial neural network models, statistical models, etc.

- techniques for discretisation of the problem domain, i.e. regular/irregular, structured/unstructured grids, etc.

- numerical approach for approximating partial differential equations, i.e. finite difference, finite element and finite volume methods

- numerical scheme for solving partial differential equations using explicit or implicit schemes, e.g. Euler scheme, Crank-Nicholson scheme, Godunov's scheme, etc.

- direct and iterative algorithms for solution of equation systems, e.g. Gauss elimination, LU decomposition, conjugate gradient method, biconjugate gradient methods, etc.

- application to steady and unsteady problems

- application to one-, two- or three-dimensional problems

To take the example of shallow water flow models, a large number of one-, two- and three-dimensional simulation models are available to simulate such flow, e.g. HEC-RAS, Mike suite of models, ISIS, Telemac, SOBEK, TUFLOW, etc. (Néelz and Pender, 2007). Similarly, for simulating subsurface flow, modellers have the option of using FEFLOW, MODFLOW, Hydrus among other similar models. A large number of simulation models are also available for urban drainage networks, water supply networks, water quality monitoring, etc. The current work makes use of HMS (Busse et al., 2012, Simons et al., 2014), developed at the Technische Universität Berlin and DuMu^{X} (Flemisch et al., 2011) developed at the Stuttgart University for simulating overland and subsurface flow respectively.

2.1.7 GIS models

The term GIS stands for Geographic Information System. It can be defined as a system that allows one 'to view, understand, question, interpret, and visualize [geographic] data in many ways that reveal relationships, patterns, and trends in the form of maps, globes, reports, and charts' (ESRI, 2013c). GIS finds application in a large number of engineering fields and is a handy tool for working with any form of geospatial related data. Some examples of GIS applications include online cartographic applications, analysing geospatial related information, e.g. calculation of the shortest or the quickest route between two points using traffic and road network data, assessing the impact of human activities such as mining, assessment of risk from natural hazards, calculation of insurance costs, etc. Applications of GIS in hydroinformatics include:

- analysis of geographic data, e.g. delineation of river catchments, calculation of flow paths, etc.
- spatial interpolation of data, e.g. precipitation, temperature or wind distribution over an area
- displaying results with spatial information from other hydroinformatic models, e.g. flood risk analysis from the results of overland flow models, etc.
- as the software environment for developing and running simulation models, e.g. SWAT (Texas A&M University, 2015)

- providing information such as results from hydroinformatic models in the form of interactive maps, time series, etc. through web-services

ArcGIS by ESRI (ESRI, 2015) is arguably the best-known example of a desktop-based GIS application. GRASS GIS (OSGeo, 2015) and Quantum GIS (QGIS Community, 2015) are open source and cross-platform GIS applications that are also associated with the Open Source Geospatial Foundation (OSGeo). MapWindow is another open source GIS application for Windows (MapWindow GIS, 2015). Google Earth (Google, 2015) is a cross-platform desktop application that allows viewing of geographic information such as maps, spatial information formatted in KML (OGC, 2008), satellite imagery, etc. This information is overlaid on a virtual globe, which the user can interact with for viewing different types of spatial information for different locations on the earth at different zoom levels.

GIS applications are not just limited to the desktop. A number of online mapping services such as OpenStreetMap (2015), Google Maps (Google, 2015), Bing Maps (Microsoft, 2015), etc. are examples of online GIS applications for viewing cartographic information, satellite and aerial imagery, calculating routes between different locations, etc. Web-services standardised by the Open Geospatial Consortium (OGC), such as the Web Map Service (WMS), Web Feature Service (WFS), Web Coverage Service (WCS) (OGC, 2015a) allow requesting and publishing of GIS data across networked computers and over the internet as well. Both ESRI and Quantum GIS support these web services through their products, ArcGIS for Desktop (ESRI, 2013a), ArcGIS for Server (ESRI, 2013b) and Quantum GIS (QGIS Community, 2013). Google earth also allows viewing information from WMS services. A number of Spatial Database Infrastructure services such as INSPIRE (European Union, 2007d), Geoportal (2015) and Geoportal Brandenburg (2015) make use of these web services for providing access to spatial data.

2.2 Software technologies

In this section, software technologies relevant for model coupling in hydroinformatics that form the foundation on which the hydroinformatic models have been developed are discussed. These include the programming languages for writing software, markup languages for describing information along with its semantics, databases for storing large datasets, network services for communicating between remote machines, etc.

2.2.1 Object-oriented programming languages

Programming languages can be divided into different categories based on the approach they use for structuring programmes e.g. procedural languages such as C, object-oriented programming languages such as Java and C++, etc. In this section, the object-oriented programming paradigm is described. Other programming paradigms have been left out from this section because they haven't been used in the current work.

Object-oriented programming is a programming paradigm that tries to abstract software concepts into entities, known as objects, that can be utilised in multiple contexts. Objects in object-oriented programming are analogous to real-world objects. Similar to real world objects, they have a state and a behaviour. Taking a microwave as an example of a real world object, the state of the microwave describes whether or not it is currently operating, the remaining time for the current operation, etc. Its behaviour is provided by various buttons and dials that allow adjusting its energy output, operational time, etc. An instance of a computational grid for a simulation model in the field of hydroinformatics is an example of a object from object-oriented programming. The state of the grid describes its geometric entities such as computational nodes, edges, faces, cells, etc. Its behaviour describes the functionality for adding or removing geometric entities to the grid, moving nodes, finding neighbours of a particular entity adapting the grid according to changes in the simulation conditions, etc. Some important object-oriented programming concepts include:

- **Encapsulation** describes the concept of limiting access to both the instance variables and its methods. This is done with the aim of achieving greater control over programme execution and reducing bugs, e.g. limiting access to the coordinates of a grid's nodes through methods designed for this purpose or ensuring that lengths, areas, volumes, etc. for all the geometric entities of the grid are properly updated.

- **Inheritance** describes the process of extending the functionality of a class by deriving more specialised classes that inherit the behaviour of the parent class, e.g. a regular structured grid may be a specialisation of a generalised grid providing additional functionality for accessing grid elements through their indices in the grid.

- **Polymorphism** describes the concept where an object may have a different

behaviour based on the context in which it used, e.g. the flux over a cell's boundary for a simulation model is calculated differently for one-, two- and three dimensional cells.

The three most important object-oriented programming languages are arguably C++, Java and C#. A large number of other languages such as Ruby, Perl, Python, Scala, etc. are also either based entirely on object-oriented concepts or support the object-oriented programming paradigm.

2.2.2 Markup languages

Markup languages provide a way to describe the semantic information within a document through the means of annotations that are clearly distinguishable from the content of the document itself (Wikipedia, 2015f). Many different markup languages are available for digital documents, e.g. Markdown (Gruber, 2004), SGML (ISO, 2005), Well Known Text (WKT), Well Known Binary (WKB) (OGC, 2011), TEX (TeX Users Group, 2015), etc. On the world wide web as well, markup languages are used extensively for authoring webpages, e.g. using HTML, XHTML (W3C, 2014), wiki articles on websites such as Wikipedia and so on.

The Extensible Markup Language, or XML for short (W3C, 2008), is a very popular general-purpose markup language. XML, itself based on the Standard Generalized Markup Language (SGML), forms the basis of a large number of other markup languages. It provides a generalised syntax for describing the semantic information for a digital document through the use of so called tags and attributes. Documents written in the XML format are not only human-readable, but can also be parsed by a large number of software tools, e.g. the Document Object Model (DOM), XQuery, XPath, XSLT, etc. This makes it a very convenient format for exchanging information among software applications.

A large number of markup languages are based on XML. These include DocBook (OASIS, 2009) for technical documents, Geography Markup Language (GML) for geographic information, Mathematical Markup Language (MathML) for mathematical equations, X3D for representing and communicating 3D scenes and objects (Web3D Consortium, 2015), etc. Some important examples of XML based markup languages in hydroinformatics are SensorML (OGC, 2015b) for sensor data and metadata, WaterML (OGC, 2015c) and UsgsHydroML (USGS, 2012) for describing hyrdological data, GroundWater Markup Language (GWML) (Boisvert and Brodaric,

2012), etc. Two of these, SensorML and WaterML are described in the following sections.

SensorML

SensorML is a markup language for describing sensor data and the related metadata. It makes it possible to (OGC, 2015b):

- discover sensors through the publication of metadata that describe the sensors
- geolocating the sensors
- describe steps undertaken in the processing of data
- have an on-demand processing of observed data, development of auto-configuring sensor networks
- develop autonomous sensor networks in which sensors can publish alerts and tasks to which other sensors can subscribe and react

WaterML

WaterML is a markup language for the representation of hydrological observation data with a specific focus on time series structures. Like SensorML, there are provisions for describing metadata, location description, steps undertaken to arrive at the data, etc. in WaterML (OGC, 2012, 2015c). These standards thus make it easy to exchange information between various software tools and can be used in scenarios such as:

- discovering measurement stations over the internet by searching for specific keywords
- displaying locations of measurement stations, domains of laboratory experiments or numerical simulations on a map along with extensive information about the sensors installed in the field or the physical or simulation model
- displaying observation data or modelling results in an interactive mode in a browser
- linking sensors or simulation tools together to foom a network where the different components are able to freely exchange information with each other

2.2.3 Databases

A database, similar to an information model (see section 2.1.2), can be defined as a structured collection of data. Databases may be based on a variety of different paradigms. Some of these are described below:

- **Relational databases** are based on the set theory in mathematics and are implemented as two-dimensional tables with rows and columns (Redmond et al., 2012), known as *relation*s (Codd, 1990). Interacting with the database is done through queries written in the Structured Query Language (SQL). Relational databases support simple data types such as strings, numbers, dates, etc. (Redmond et al., 2012). SQLite, Microsoft Access, MySQL, MariaDB, Oracle, etc. are examples of relational database management systems. A more complete list of relational database managements systems can be found on Wikipedia (2015l).

- **Object-oriented databases** also known as object-databases organise data following object-oriented principles. These include concepts such as complex data types, encapsulation, inheritance, etc. (Bancilhon et al., 1992) (see also section 2.2.1). Most object-oriented databases use the Object Query Language (OQL) as their query language. A list of object-oriented database management systems can be found on Wikipedia (2015h).

- **Object-relational databases** are hybrid databases that add object-oriented features to the relational databases. Such features include:

  - structured types for values in addition to simple types such as numbers, strings, etc. (Garcia-Molina et al., 2009)
  - methods associated with relationships (Garcia-Molina et al., 2009)

  PostgreSQL is an example of an object-relational database. A complete list of relational database management systems is available on Wikipedia (2015i).

- **Hierarchical databases** are databases where data are stored in tree-like structures with data represented as a collection of records and relationships between data represented as links (Kamfonas, 1992, Silberschatz et al., 2010). Information Management System (IBM, 2012) by IBM is an example of a hierarchical database.

- **Network databases** represent data as a collection of records and relationships between data as links. As opposed to hierarchical databases, the records can be organised as complex graphs, rather than just tree-like structures possible in hierarchical databases (Silberschatz et al., 2010). A list of databases using the network model can be found on Wikipedia (2015g).

- **Columnar databases** are column-oriented databases where the data from individual columns are stored together. This is in contrast to the row based storage approach of the relational database management systems (Redmond et al., 2012). Examples of columnar databases include IBM DB2 and MonetDB.

- **Key-value databases** have a very simple structure and store data in the form of key-value pairs (Redmond et al., 2012). Berkeley DB is an example of a key-value database management system.

- **NoSQL databases** are a class of databases that do not follow the traditional relational database model and don't use SQL as the query language (Vaish, 2013). Although NoSQL is generally interpreted as *Not only SQL*, it does not exclude the possibility of using SQL for interacting with the database. NoSQL databases can belong to any of the previously mentioned database classes.

Although not strictly a database, PostGIS is an important piece of software that requires a special mention in this section because of its importance to GIS models. It is a spatial database extender for PostgreSQL that adds support for geographic objects allowing location queries to be run in SQL (PostGIS, 2015b). A large number of desktop and web based GIS applications including ArcGIS, Quantum GIS, GRASS GIS, Mapinfo, etc. make use of PostGIS (Wikipedia, 2015k). Oracle Spatial is another database with support for geographic objects that can be used for GIS applications.

2.2.4 Network and communication services

An increasing number of devices today have networking capabilities and are able to communicate over the internet. As a result, it is possible to have scenarios where simulation models pull information directly from field instruments, update their simulation results and make them available to other models. One method of achieving connectivity among models is through web services. By making use of web services, models can expose some of their functionality to the public network which can

then be accessed by the clients of the web services over a network connection. Web services can play a useful role in model coupling by making it possible to couple hydroinformatic models running on separate machines over a networked connection.

Remote Procedure Calls (RPC) involve invoking methods on a remotely running piece of software, possibly over a networked connection. RPC, in association with technologies such as XML, forms a popular method for implementing web services. Three RPC standards, XML-RPC, SOAP and JSON-RPC are briefly described in the following sections.

XML-RPC

The XML-RPC specification is an RPC protocol which uses XML as encoding and HTTP (Leach et al., 1999) as the transport mechanism (UserLand Software, 2003). The *modus operandi* for performing RPC using XML-RPC can be summarised as follows:

- A server capable of receiving XML-RPC requests is set up and made accessible over the internet.
- A software process makes a request to the server in the form of an HTTP-POST (Leach et al., 1999) request with an XML payload containing information such as the name of the remote subroutine or method to call and the parameters to be passed to this subroutine or method. Listing 2.1 shows an example of such a request. Here lines 1–4 are the HTTP headers for the client's request, with the rest of the message being the XML-RPC payload.
- The server executes the request and returns either the result of the procedure call or an error message if it cannot fulfil the request. The response is again encoded as XML. Listing 2.2 shows an example of a response from the server after the successful completion of the XML-RPC request shown in listing 2.1. Here, lines 1–4 are the HTTP headers for the server's response and the rest of the message is made up of the result of the RPC request.

Listing 2.1: Example of an XML-RPC request

```
POST /tensor/unit-converter HTTP/1.1
Host: example.org
Content-Type: text/xml
Content-length: 447

<?xml version="1.0"?>
<methodCall>
   <methodName>tensor.convertToSI</methodName>
   <params>
      <param>
         <struct>
            <member>
               <name>magnitude</name>
               <value><double>8.4</double></value>
            </member>
            <member>
               <name>units</name>
               <value><string>feet</string></value>
            </member>
         </struct>
      </param>
   </params>
</methodCall>
```

Listing 2.2: Example of an XML-RPC response

```
HTTP/1.1 200 OK
Connection: close
Content-Type: text/xml
Content-length: 413

<?xml version="1.0"?>
<methodResponse>
   <params>
      <param>
         <struct>
            <member>
               <name>magnitude</name>
```

```
13                <value><double>2.5603</double></value>
14             </member>
15             <member>
16                <name>units</name>
17                <value><string>metres</string></value>
18             </member>
19          </struct>
20       </param>
21    </params>
22 </methodResponse>
```

The XML-RPC standard specification also has elementary support for some basic data types that are commonly used in many programming languages. These are (Laurent et al., 2001, UserLand Software, 2003):

- null/nil type (the `<nil>` tag)
- boolean types (the `<boolean>` tag)
- numerical data types that are either integers (`<int>` or `<i4>` tags) or double-precision floating-point types (`<double>` tag)
- strings (the `<string>` tag)
- date-time in the ISO 8601 format (ISO, 2015) (the `<dateTime.iso8601>` tag)
- base 64 encoded (Josefsson, 2006) binary data (the `<base64>` tag)
- arrays for multiple values of the same type (the `<array>` tag)
- more complex data structures which contain one or more of the above data types and may be used to mimic objects (the `<struct>` tag)

The XML-RPC specification has been implemented in a large number of programming languages with libraries available for C/C++, Java, Perl, PHP, Ruby, Python among others. A list of XML-RPC implementations for various languages can be found on Wikipedia (2015p).

SOAP

SOAP is another specification for implementing web services. SOAP was originally an acronym for *Simple Object Access Protocol* (Chappell and Jewell, 2002, W3C,

2007a) and similar to XML-RPC, it makes use of XML for encoding messages. As opposed to XML-RPC, however, SOAP isn't limited to using HTML as the transport mechanism and SOAP messages can be transmitted using other protocols such as SMTP (Klensin, 2008) as well. Although SOAP can be used to perform remote procedure calls, this is, by design, not its primary purpose. The basic type of document exchange over the SOAP protocol is unidirectional with RPC being just a specialized case consisting of multiple such unidirectional exchanges being combined together in a sort of a request-response mechanism (Chappell and Jewell, 2002).

Listing 2.3: Example of a SOAP request over HTTP

```
1  POST /tensor/unit-converter HTTP/1.1
2  Host: localhost
3  Content-Type: text/xml
4  Content-length: 360
5
6  <?xml version="1.0"?>
7  <soap:Envelope
8        xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
9     <soap:Header>
10    </soap:Header>
11    <soap:Body>
12       <tensor:convertToSI
13             xmlns:tensor="http://localhost/tensor">
14          <tensor:magnitude>8.4</tensor:magnitude>
15          <tensor:units>feet</tensor:units>
16       </tensor:convertToSI>
17    </soap:Body>
18 </soap:Envelope>
```

Listing 2.3 shows an example of a SOAP message used to send an RPC request over HTTP. Here, lines 1–4 are the HTTP headers, with the rest of the listing showing the SOAP payload. Simlar to this example, each SOAP message is encoded using XML and consists of a SOAP body and an optional SOAP header. The SOAP body and the SOAP header are enclosed within a SOAP envelope and each of them can contain multiple XML blocks (Chappell and Jewell, 2002). In listing 2.3, lines 7, 9 and 11 show the beginnings of the `Envelope`, `Header` and `Body` elements respectively. Lines 11–14 show the SOAP block, enclosed within the `Body` element. This is the actual remote procedure call request enclosed in this SOAP message. It contains

information such as the remote method to be called and the parameters to be passed to this method. The SOAP header is empty in this particular case but SOAP servers and clients can decide upon and enclose any additional information, e.g. metadata about the content of the `Body` element, within the `Header` element (Chappell and Jewell, 2002).

JSON-RPC

JSON-RPC is an RPC protocol which is quite similar to XML-RPC, but uses the *JavaScript Object Notation* (JSON) (Crockford, 2006) instead of XML to represent messages and server responses. JSON-RPC calls, similar to SOAP calls, can be made over several protocols including HTTP (JSON-RPC Working Group, 2010). The following basic data-types, that are defined in the JSON standard, are also supported by JSON-RPC (Crockford, 2006, JSON, 2015):

- Null
- Booleans
- Numbers (integers and floating-point types)
- Strings
- Objects
- Arrays

Listing 2.4: Example of an JSON-RPC request

```
{
   "jsonrpc": "2.0",
   "method": "convertToSI",
   "params": {
      "magnitude": 8.4,
      "units": "feet"
   },
   "id": 1
}
```

Listings 2.4 and 2.5 show the JSON-RPC equivalents of the RPC request and response shown earlier for XML-RPC in listings 2.1 and 2.2 respectively. Implementations of the JSON-RPC standard also exist for a large number of programming languages such as Javascript, Java, C++, Perl, Ruby, Python, PHP, etc. (Wikipedia, 2015c).

Listing 2.5: Example of an JSON-RPC response

```
{
   "jsonrpc": "2.0",
   "result": {
      "magnitude": 2.5603,
      "units": "metres"
   },
   "id": 1
}
```

2.2.5 Software libraries

Any piece of software that is sufficiently complex can be broken down into smaller units. These units are usually reusable across different software and as a result can be abstracted into separate entities called software libraries. When developing a new piece of software, the developer then has the choice to link his software to these libraries instead of reimplementing their functionality on his own. For hydroinformatic models, functionality that is shared among different models and can be abstracted into software libraries include, for example, equation-system solvers, grid generators, etc. Advantages of using software libraries include:

- reduction in redundancy where the same functionality is implemented multiple times
- reduction in duplication of effort required for development of software, debugging, fixing bugs, etc.
- sharing of benefits such as performance improvements, bug fixes, etc. by every piece of software using the library

Software libraries that are useful in the development of hydroinformatic models can be broadly divided into the following two categories:

General-purpose software libraries

The general-purpose software libraries have a scope that is broader than just hydroinformatic models. They include libraries providing functionalities such as equation system solvers, visualisation tools, etc.

The Commons Math (Apache Commons, 2015) library by Apache Commons is an example of a general purpose mathematics library for Java that provides a collection of mathematical functions such as iterative equation solvers, interpolation functions, geometric operations, statistical functions, etc.

R is a programming language that provides a lot of useful statistical functions as part of its standard library (The R Foundation for Statistical Computing, 2015). Programmes written in other programming languages can also leverage the functionality provided by R through libraries such as rJava (rJava, 2015) for Java.

Libraries such as NumPy (NumPy, 2015), matplotlib (The matplotlib development team, 2015) and RPy (RPy, 2012) software libraries providing mathematical functions, plotting tools and R bindings respectively for Python. Similar solutions also exist for other programming languages.

Software libraries also exist for working with different file formats. Libraries such as JDOM (JDOM, 2015) provide functionality that is useful for working with XML documents, e.g. parsing of XML documents, outputting XML documents, performing XPath queries or XSL transformations, etc.

Inter-Process Communication (IPC) is a mechanism for software processes to communicate with each other. A number of IPC mechanisms exist for different operating systems such as reading and writing files, pipes, sockets, semaphores, Remote Procedure Calls (RPC), etc. (Bezroukov, 2014, Microsoft, 2015, The Linux Documentation Project, 1996, Wikipedia, 2015b). As already mentioned in section 2.2.4, several libraries provide functionality for performing RPC, including XML-RPC, SOAP and JSON-RPC. Examples include XML-RPC implementations for C and C++ (for C and C++, 2015), Apache XML-RPC (Foundation, 2010) for Java, jsonrpc4j (Dilley, 2015) a JSON-RPC implementation for Java, JsonRpc-Cpp (JsonRpc-Cpp, 2011) a JSON-RPC implementation for C++, etc.

As mentioned in section 2.2.3, PostGIS is a database extender for PostgreSQL, that adds support for geographic objects to PostgreSQL. The geographic data stored in a database with PostGIS support can be accessed and queried like other database models using SQL statements. GIS applications such as GRASS GIS (OSGeo, 2015) provide interfaces for GIS operations that can be utilised by applications such as

Quantum GIS (QGIS Community, 2015). Applications such as ArcGIS (ESRI, 2015), Geoserver (Foundation, 2015a) and Mapbender (2015) provide access to spatial data in databases such as PostGIS (PostGIS, 2015b) through the use of web services. Together with JavaScript based mapping libraries such as OpenLayers (2015) and GeoExt (GeoExt community, 2015), they allow the creation of web-based mapping applications like the one presented in section 5.4.

A number of libraries for the creation of GUI for different platforms developed in various programming languages are available. Some of the best known examples of cross-platform libraries for the creation of GUIs include Qt (The Qt Company, 2015) and GTK+ (2015). Libraries such as Swing (Fowler, 2013) and JavaFX (Oracle, 2014) allow for the creation of GUIs for Java applications and in the case of JavaFX, for web applications where the server runs using Java technologies. JavaScript libraries such as Ext JS Sencha (2015) allow the creation of GUIs for interactive web applications. The GeoExt library mentioned in the previous paragraph is based on the Ext library.

Numerical modelling toolboxes

Numerical modelling toolboxes are software libraries that have been designed specifically with hydroinformatic models in mind. They provide functionality such as grid generators, equation-system solvers, file input/output in certain formats, etc. This type of functionality is useful in the development of models such as simulation models performing time based simulations using grid based spatial discretisation.

An example of an equation-system solver is the BLAS (Basic Linear Algebra Subprograms) library. BLAS is a 'specification of a set of kernel routines for linear algebra' (Netlib, 1979a). A number of other equation solver libraries such as LINPACK (Netlib, 1979b) and LAPACK (Anderson et al., 1999, Netlib, 2013) are based on BLAS. LAPACK implementations are available for a variety of different platforms and a number of mathematical software such as Matlab, Octave, Maple, R, etc. make use of the LAPACK library (Netlib, 2015).

For grid based simulations, e.g. for finite element modelling, METIS provides a 'set of serial programs for partitioning graphs, partitioning finite element meshes, and producing fill reducing orderings for sparse matrices' (Karypis and Kumar, 1998, Karypis Lab, 2015). ParMETIS is an 'MPI based parallel library that extends the functionality provided by METIS and includes routines that are especially suited for parallel AMR [Adaptive Mesh Refinement] computations and large scale

numerical simulations' (Karypis Lab, 2015).

Higher level numerical modelling toolboxes provide functionality that is based on the tools mentioned above. Examples of such libraries include OpenFOAM (OpenFOAM Foundation, 2015) and DUNE (Bastian et al., 2008, DUNE, 2015), which can be used for developing simulation models in C++. These toolboxes provide the functionality for working with different grid types, iterative equation solvers, parallelisation for more efficient computation, etc. It is therefore possible to put together components provided by these toolboxes to construct one's own hydroinformatic model.

2.3 Model coupling

Coupling of models involves combining multiple models in a way where models can share and exchange information in a controlled manner, so that output from one model can be used as a meaningful input for another model. This results in a set-up where models run sequentially, i.e. one after the other, or simultaneously while this information exchange and/or sharing takes place. Within interdisciplinary hydroinformatics related projects, it is necessary to couple different kinds of models such as simulation models, laboratory models, visualisation models, etc. For this purpose, different approaches for model coupling have been developed over time. These approaches can be broadly divided into two main categories: tight coupling and loose coupling of models. These are described in more details in the following sections.

2.3.1 Tight coupling

Tight coupling of models can be described as an approach when all the coupled models are integrated within a single software package and aren't independent of each other any more. A good example of a tightly-coupled model in hydroinformatics is the *System Hydrologique Europeen* (SHE) (Abbott et al., 1986) which has separate modules for simulating different parts of the hydrological cycle such as precipitation, runoff, evaporation, transpiration, infiltration, etc. The tightly coupled modelling approach has the advantage that it 'provides complete control over the process representations and data structures within all parts of the model' (Goodall et al., 2011). As a result, the interactions between different modules can be optimised by improving the mathematical and physical model that forms the basis of the code base and by optimising the information exchange between the individual models. However, this approach has the disadvantage that it is inflexible and adding additional mod-

els to an existing set-up is a cumbersome process, especially if they don't confirm to the existing conventions such as data structures and semantics that already exist in the coupled set-up (Goodall et al., 2011). Furthermore, such modifications are impossible without access to the source code of the model. Some examples of tightly coupled models for modelling surface-subsurface flow include MODFLOW, SWAT, TOPMODEL, Hydrosphere, etc. (Kolditz et al., 2008). Since tight coupling of models is not the focus of the current work, it isn't described in further details here.

2.3.2 Loose coupling

The alternative coupling strategy to tight-coupling is the loose coupling of models. Here individual models continue to exist as independent entities. Coupling is achieved by sharing and exchanging information among coupled models through predefined interfaces. The coupled models may be located on the same machine and interact through mechanisms such as file exchange, sockets or using platform specific features such as UNIX pipes. Models that are running on separate machines can also be loosely coupled over a network connection via internet, telecommunications networks, etc.

Loose coupling has the advantage that individual models can be developed and maintained independent of each other. It also provides the flexibility to change the combination of the coupled models by adding, removing or replacing individual models from the coupled set-up. However, for coupling to be possible, all models must implement some common interfaces. Also, the flexibility provided by loose coupling comes at the cost of performance because of the inherent latencies when communicating with an external piece of software and also because it is impossible to predict every set-up in which a model will be coupled and to optimise for them. We now take a more detailed look at two software standards that have been developed for loose coupling models.

Object Modeling System

Object Modeling System (OMS) is a framework for environmental model development that is based on the concept of *Inversion of Control* (David et al., 2013) in programming. Inversion of Control can be explained in terms of two main types of entities:

- the main execution branch which is responsible for the business logic and the execution of the programme

- individual modules that implement or override some of the classes used by the main execution branch to provide some sort of concrete functionality

This approach allows for the independent implementation and maintenance of the framework providing the main execution control and of modules that carry out tasks such as pulling data from field instruments, connecting to databases, carrying out computational/modelling tasks, outputting simulation results, etc. Using an inversion of control framework such as OMS, the individual modules can be appropriately chosen and varied according to the modelling requirements.

According to the developers of OMS, it 'is a framework for environmental model development that provides a consistent and efficient way to:

- 'create science simulation components
- 'develop, parameterize, and evaluate environmental models and modify/adjust them as science advances, and
- 're-purpose environmental models for emerging customer requirements' (David et al., 2013).

OMS is an open source project which has been developed using Java and which uses features of the Java programming language such as *interface*s and *annotation*s to achieve the goals mentioned above. The latest version of OMS is OMS3 (David et al., 2013).

OpenMI

OpenMI (Gregersen et al., 2007) is a widely used standard for linking of models in the field of hydroinformatics. The name OpenMI stands for Open Modelling Interface, a standard which is maintained by the OpenMI consortium. It has its roots in the HarmonIT project (OpenMI, 2012a) which was led by the UK's Centre for Ecology and Hydrology with the support from the European Commission. Fourteen different academic and industrial organisations from seven different countries were part of the project. The group aimed to facilitate integrated catchment management for modelling all the processes in an entire river catchment by defining a standard for linking of software components that are capable of modelling the individual processes. As part of the project the group worked towards identifying the requirements for such a linking of software components, specifying the domain boundaries for the linking and to develop and test a set of software tools that make this linking possible.

The work done as part of the project resulted in the version 1.0 of the OpenMI standard, with the release of a .NET implementation which meant that it could be used for linking software components running under the Microsoft Windows operating system (Fortune et al., 2008, OpenMI, 2012a,d).

The HarmonIT project was followed by the OpenMI-Life project (OpenMI, 2012b,c) for transforming the research output of the HarmonIT project into an operational standard. The project's objective of setting up a structure for the support, maintenance and dissemination of the OpenMI standard led to the formation of the OpenMI consortium. It also led to the release of the updated version of the standard, most importantly the version 1.4, which for a long time was the only officially supported version (OpenMI, 2012c).

The latest version of OpenMI is 2.0. Work on this version started as part of the OpenMI-Life project, but the actual release was after the end of the project in December 2010. Until version 1.4, the OpenMI standard was mainly developed for time based linkable components such as simulation tools for unsteady problems. Version 2.0 reduces this time dependence and enables to link components providing data that vary in space or time or both (OpenMI, 2012c, The OpenMI Association Technical Committee, 2010b). It is therefore possible to link components such as geo-databases using OpenMI 2.0.

Apart from C#, the OpenMI standard is also available for Java with the first Java version being made available for OpenMI 1.4 (OpenMI, 2012c). As of OpenMI 2.0, software components that are written in the two different programming languages still cannot interact using the standard (The OpenMI Association Technical Committee, 2010a).

Linking models with OpenMI : The OpenMI standard consists of a set of interfaces for linking software components. For coupling a model using the standard, it must implement a minimum set of these interfaces. Any model that implements this minimum set of interfaces becomes a so called *linkable component*. All communication between different linkable components takes place through a single interface. Models implementing the OpenMI standard therefore have the freedom to make any modifications to the software itself as long as they are able to interact through that interface (Fortune et al., 2008, The OpenMI Association Technical Committee, 2010a).

Data exchange between models linked using OpenMI takes place through the use of so called *exchange item*s. OpenMI compliant models define the exchange items

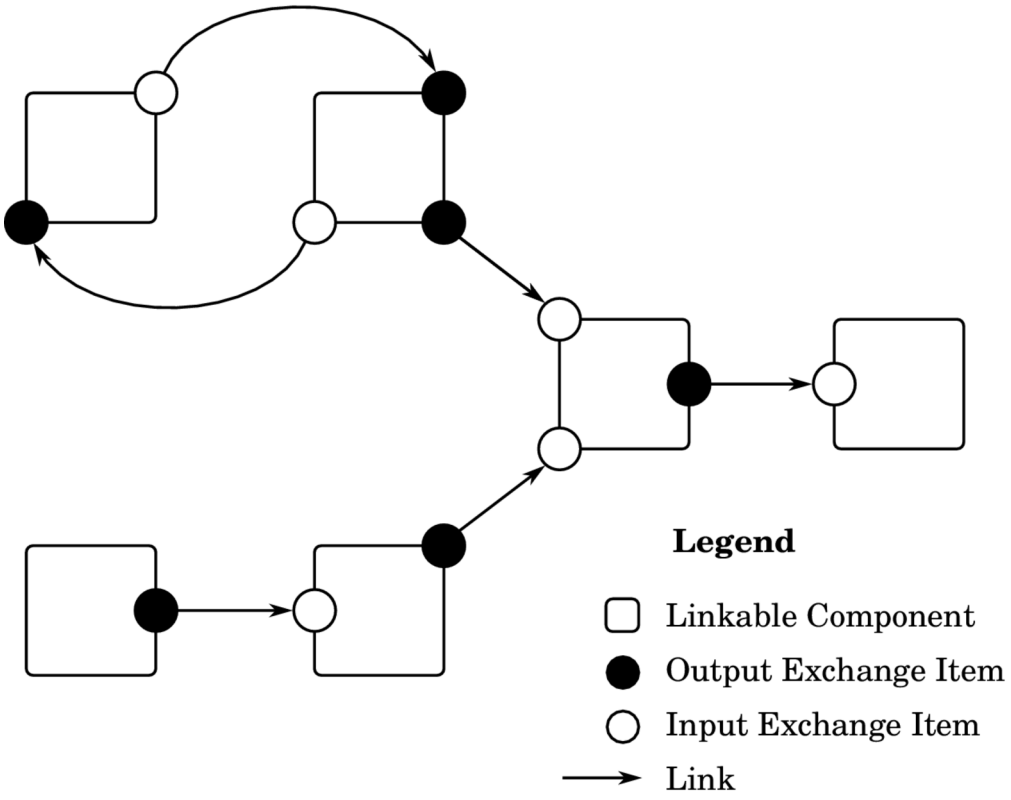

Figure 2.1: Schematic diagram of models linked using OpenMI

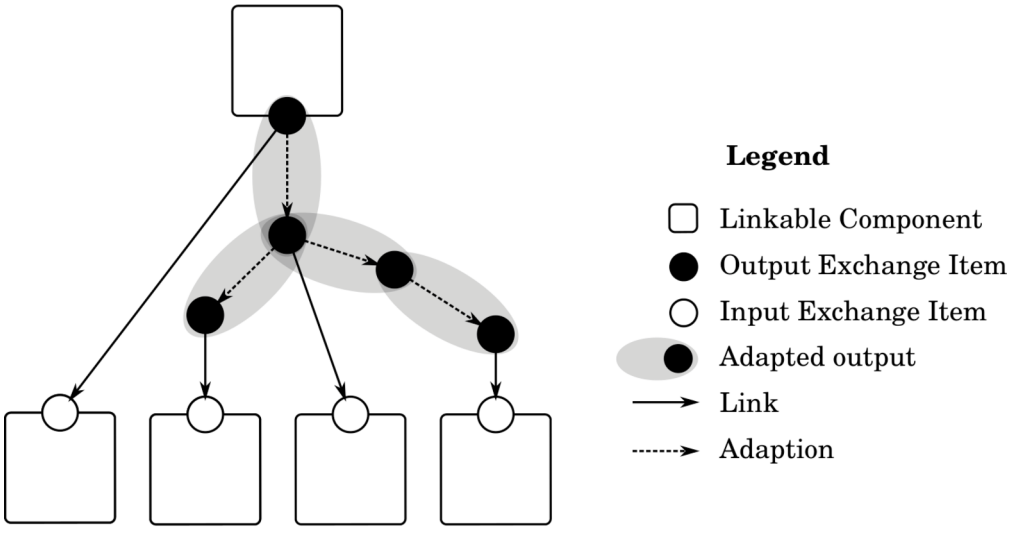

Figure 2.2: Schematic diagram of OpenMI adapted output

that they can accept as an input and those that they can provide as output to other linkable components. Since a model can define both input and output items, it can simultaneously act as both a data producer and a consumer. To make the information about the input and output exchange items discoverable, models can publish it along with any other information necessary for successful linking of components in an XML document known as the OMI file (Fortune et al., 2008, The OpenMI Association Technical Committee, 2010a).

Figure 2.1 shows a schematic diagram of linkable components linked using the OpenMI standard. After the link has been set up, the linkable components can begin exchanging data. Data exchange happens when one linkable component makes a request to another one at the other end of the link. The data exchange is therefore defined to be *pull based*. Models are able to save their state, which may be useful for situations where, for example, iterative solutions may be required (Fortune et al., 2008, The OpenMI Association Technical Committee, 2010a). OpenMI makes use of the following metadata for exchanging data:

- **What**: These metadata can either be quantities, i.e. one or more *value*s as defined in the standard, along with their units or qualities and an enumeration of the possible values for that quality (The OpenMI Association Technical Committee, 2010a).

- **Where**: *Element*s, as defined in the standard, where information exchange takes place. Elements contain both geometric and topological information and may or may not be georeferenced (The OpenMI Association Technical Committee, 2010a).

- **When**: This is either a single point along the time axis, i.e. a time stamp or a range, i.e. time span defined in Modified Julian Date (The OpenMI Association Technical Committee, 2010a).

- **How**: Any additional steps that are undertaken to prepare the data before exchange, e.g. interpolation, extrapolation, aggregation, etc. The linkable component providing the output is responsible for any such adaptations that may be required to provide the requested items (The OpenMI Association Technical Committee, 2010a).

OpenMI 2.0 introduced the concept of *Adapted Outputs* (see figure 2.2). Adapted output items are specialisations of the *output item*s (defined in the standard)

that linkable components provide. They take the data from an output and modify it through operations such as unit conversion, interpolation, extrapolation, etc. in order to adapt it to the requirements of the linkable component requesting the data (The OpenMI Association Technical Committee, 2010a). Although they are called '*adapters*' in the standard, adapted output items actually resemble *Decorator*s from the Decorator design pattern (Freeman et al., 2004, Gamma et al., 1994) in both their design and functionality, since they take the output from another item, decorate it by performing some operations on that output and provide the result, with the result itself also being an output item.

In order to make models implementing the OpenMI standard easily discoverable, it is possible to register them with the OpenMI association. Such models are then known as *OpenMI compliant component*s (OpenMI, 2012e). As of January 2014, 24 registered OpenMI compliant components are listed on the OpenMI website (OpenMI, 2012f).

2.4 Evaluation of the state-of-the-art

In this chapter, the suitability of the existing technologies for coupling of models is evaluated. In section 2.4.1, we first take a look at the criteria that are used in this evaluation and later on in section 2.4.2, we discuss the existing pieces of technologies that are most suitable for the development of a framework for coupling of models in hydroinformatics.

2.4.1 Requirements for model coupling

For any sort of development work, it is first important to select proper resources for that task. In order to evaluate whether or not an existing piece of technology is suitable for a particular task, it is necessary to first define the criteria for this evaluation. For the development of a model coupling framework, the following criteria have been considered to be quite important:

- **Availability** means the ease with which a resource can be acquired. If an application depends on some resource that is quite essential but is difficult to acquire, then it would lead to disruptions and to unnecessary delays during development and later during its application.

- **Dependencies** of a resource refers to the number of other resources that it

depends on in order to function. Having fewer dependencies causes fewer complications during the development, installation and application stages and is therefore desirable. Furthermore, having fewer dependencies means that a piece of software is less susceptible to performance issues that may be caused by one of the dependencies. Dependency of a piece of software may also refer to its application being restricted to a particular set of hardware or in other words, it's platform dependence.

- **Robustness** refers to the ability of a resource to perform without any faults. A special meaning of robustness also implies that the application remains available for use even under heavy load conditions.

- **Reliability** refers to the trustworthiness and consistency of results that a resource provides.

- **Efficiency** refers to the efficiency of a particular resource. In the case of software, this refers to time required to finish a particular task, the processing power or memory required, etc.

- **Familiarity** refers to the pre-existing knowledge of the developers and users with a resource. Use of technologies familiar to the developers and users of a software application reduce the time required for both development of that application and for training users to use that application.

- **Flexibility** refers to the ease with which a resource can be integrated into the existing development environment. This is of vital importance since a resource that easily fits into the existing development environment avoids unnecessary complications during integration and ensures that all components of the framework work harmoniously with each other during runtime.

- **Adaptability** refers to the ease with which a particular resource can be customised for integrating it in the existing working environment. In the case of software, it is less complicated to directly modify software for which the source code is available than developing wrappers for it in order to achieve the same functionality.

- **Support** refers to the ease with which help for using a particular resource can be acquired. This can be in the form of telephone or email support by the resource provider, user manuals, internet forums, etc. Support also refers to the

availability of other resources that make working with a particular resource easier. For software, this includes integrated development environment, software libraries, etc.

- **Cost** is another important factor that has to be taken into consideration. It is better to use resources that are available at lower costs, if not totally free of charge.

2.4.2 Technologies suitable for model coupling

Once the criteria for the selection of resources for the development of a model coupling framework have been defined, we can start to take a look at the existing pieces of technology in order to select the ones that are most appropriate for the task.

Object-oriented programming languages

The choice of languages supporting the object-oriented paradigm is quite large and includes languages such as C++, Java, C#, Objective-C, Scala, Ruby, Python, etc. Of these, C++, Java and C# are arguably the most popular ones. All of these are compiled languages, which means that they perform better than interpreted languages such as Ruby, Python, etc. However, only C++ gets compiled to native machine code. Java and C# on the other hand get compiled to bytecode which is then interpreted by a virtual machine or if required, compiled to machine code by it using just-in-time compilation. Therefore, from the perspective of performance, the differences between the three languages are not that large, with C++ arguably having a slight advantage because of being compiled to machine code.

From the point of view of requirements, Java has an advantage over most other programming languages since the Java Virtual Machine (JVM) (Lindholm et al., 2013) under which Java programmes run, has been ported to a large number of different platforms (Wikipedia, 2015e). It therefore comes really close to the ideal of *write once, run anywhere* (ComputerWeekly.com, 2002) code. On the other hand, C++ code can be described as *write once, compile anywhere* (Wikipedia, 2015o). Although C#, like Java, runs under a virtual machine, it lags behind the other two languages from the perspective of requirements, since C# code only runs on the Windows platform. However, efforts are being made by the Mono project (Mono Project, 2015) to make C# code run on other platforms.

Java has been chosen as the programming language for the development of the

prototype for a model coupling framework that is described in this work. The reason for this is that Java provides performance that is comparable to C++ and C# and Java programmes once written can usually be run on multiple platforms without modifications. Java and a number of development tools such as Eclipse (The Eclipse Foundation, 2015), Netbeans (Oracle, 2015b), etc. for developing Java applications are available free of cost. It also has a large number of free and open-source software libraries that provide functionality similar to the one already described in section 2.2.5.

Network services and communication

The three standards described earlier in section 2.2.4, XML-RPC, SOAP and JSON-RPC are quite similar to each other. There is excellent support for all three of them for a variety of programming languages. SOAP has the advantage over the other two because it is expandable through the use of custom XML schemas in the SOAP head or body. However, this functionality isn't that important in the approach for coupling of hydroinformatic models used in the current work. JSON-RPC doesn't use XML to represent information and is therefore able to do so using far lesser content size. Using JSON-RPC will therefore result in saving bandwidth. For the purpose of the current work, however, XML-RPC has been chosen for the purpose of communicating among hydroinformatic models because of prior familiarity with the technology and also because it provides all the required functionality.

Software libraries

The choice of software libraries to use depends on the programming language that is chosen for the development work. As mentioned in the previous section, out of the object oriented languages discussed in section 2.2.1, Java was chosen for the development work for the current work due to prior knowledge with the language. Apart from Java, JavaScript was used for developing a web application discussed in section 5.4.

Similar to the choice of the programming language, out of the technologies discussed in section 2.2.4, XML-RPC was chosen for communication between the coupled models and the coupling broker (see section 3.4) due to prior familiarity with that technology. Related to XML-RPC, the Apache XML-RPC library (Foundation, 2010) was used for setting up XML-RPC servers, while the Apache XML-RPC library (Foundation, 2010) and the XML-RPC library for C and C++ were used for

communicating with XML-RPC servers from Java and C++ applications respectively. The JDOM (2015) library was used for parsing and encoding XML streams.

Out of the databases discussed in section 2.2.3, PostgreSQL (The PostgreSQL Global Development Group, 2015) with PostGIS (PostGIS, 2015b) was used to set up the geodatabase for the web application discussed in section 5.4. The choice was made based on the fact that these technologies are available free of cost and there is abundant support available in the form of online documentation, forums, etc.

GIS based technologies were discussed in section 2.1.7. Out of these, GeoServer (Foundation, 2015a) and GeoExt (GeoExt community, 2015), which in turn is based on OpenLayers (2015) and (Sencha, 2015), were used for setting up the information management system discussed in section 5.4, because these technologies are available free of cost, provide extensive online documentation with examples and are based on familiar technologies discussed above, i.e. Java and JavaScript.

Chapter 3

Model coupling concept

In this chapter we take a look at a generalised concept for coupling models in hydroinformatic systems. First the goals that a framework for coupling of models tries to achieve are described. This is followed by a look at the mechanism for the model coupling. Finally, the rest of the chapter is dedicated to explaining the design of a framework that is able to achieve the goals set out in the beginning of the chapter.

3.1 Goals for model coupling

Any mechanism for coupling of hydroinformatic models should make the task of coupling easier for the coupled models. In order to do so, it should have the following properties:

- **Completeness**: Coupled models need to share or exchange information but different models may represent information differently. Therefore, a complete and generalised representation of information is desirable.

- **Autonomy**: The representation of information used by the coupling mechanism should be autonomous and capable of existing independent from that of any of the coupled models themselves.

- **Adaptability**: The ideal situation in model coupling is where the information provided by one model can be directly used by other models. In reality, this is rarely possible because of differences in space and time scales, space and time discretisations, internal data structures, etc. Therefore, the coupling frame-

work should not only be able to represent information, but also to adapt it to the requirements of behalf of the coupled models.

- **Flexibility and extensibility**: Using the coupling mechanism, it should be possible to flexibly couple a large range of hydroinformatic models such as measurement models, laboratory models, simulation models, statistical models, etc. in different combinations. It should also be possible to replace models in the coupled set-up or to add or remove models according to the coupling requirements.

- **Simplicity**: The coupling mechanism should reduce the modifications required to individual models to be able to couple them in different combinations. It should carry out the adaptation of information through operations such as scaling, mapping, etc. or more complex adaptations based on the understanding of the laws of physics for the transformation of physical state variables.

- **Communicability and mobility**: In order to couple models, it is absolutely necessary that they are able to communicate with each other to be able to share and exchange information amongst themselves. Furthermore, it is desirable for such communication to be possible among models running on different computers and different hardware architectures and not be limited to models running on just one computer, i.e. it should also be platform independent.

3.2 Model coupling approach

It is possible to implement a variety of different mechanisms for the coupling of hydroinformatic models that are able to achieve the goals mentioned in the previous section. The current work proposes a mechanism for the coupling of models that has the following features:

- The concept of tensor objects (Molkenthin, 2000, Molkenthin et al., 2009a,b) is used for representing information. Tensor objects, that are described in more details in section 3.3, provide a means to represent all the information that is necessary for coupling models and are also able to adapt to the requirements of the coupled models.

- Communication between the coupled models takes place over a network connection through XML-RPC. This way, it is possible to couple models that might

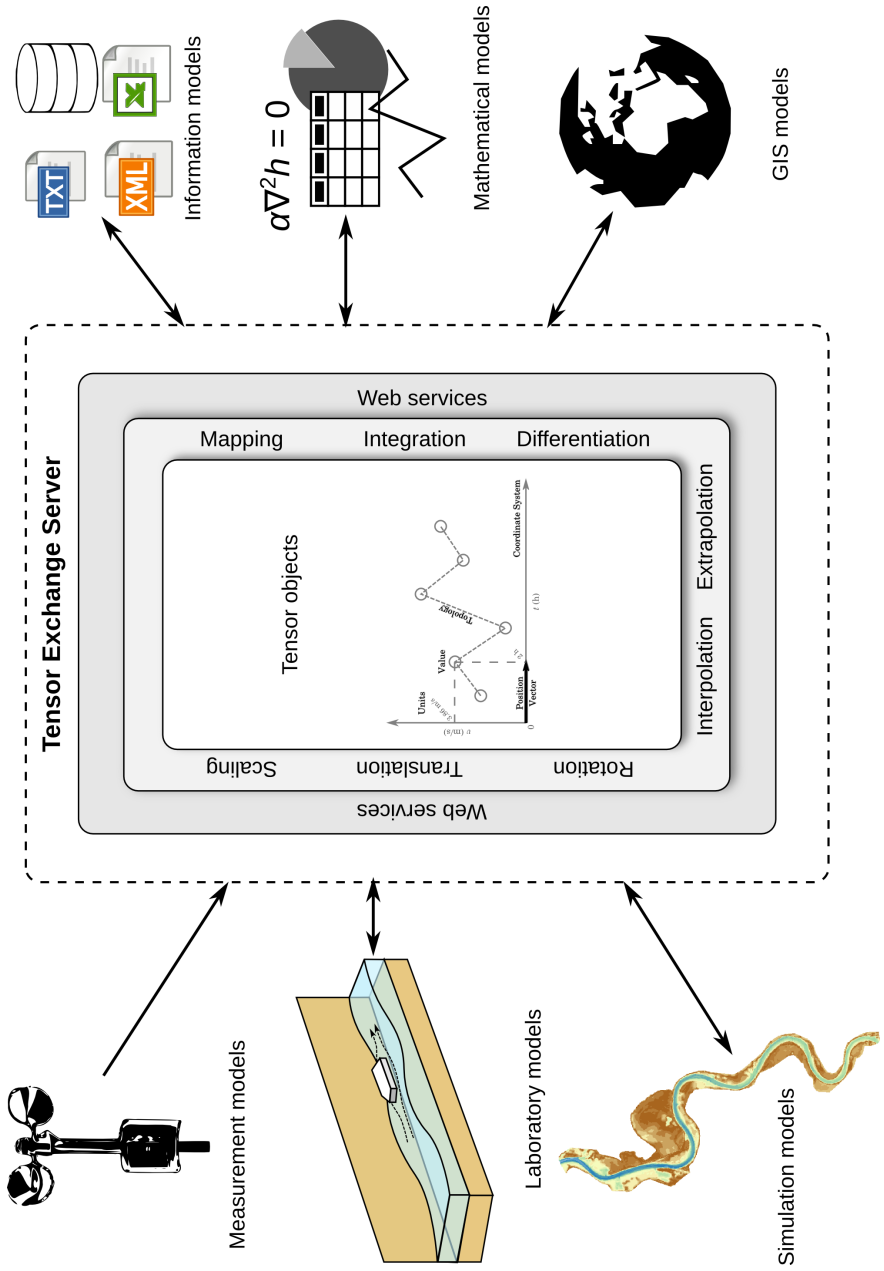

Figure 3.1: Model couplng concept based on tensor objects. File type icons from the RRZE icon set (The Regional Computing Centre of Erlangen, 2014) and the earth icon by Brightmix (2014) used under the Creative Commons (2015) licence.

be running on the same computer or over the intranet or internet. These might be different types of models, e.g. simulation models, measurement models, etc. running on different types of hardware and software, etc.

- A coupling broker is the central entity in the coupling process and brokers the communication between the coupled models.
- The coupling broker works internally with tensor objects but is able to understand and convert information from and to formats that are supported by the coupled models. It is therefore responsible for adapting the information to and from these formats. The coupling broker therefore simplifies the coupling process by eliminating the need for implementing these adaptation methods by the coupled models.
- The coupling broker also controls the coupling mechanism. In order to be able to fulfil the information requests of the coupled models, it is programmed with the ability to know the source of this information, to request or wait for this information from other coupled models, and how to adapt this information to the requirements of the model requesting that information.

The sequence of steps that are involved in the coupling process can be summarised as follows (see figure 3.2):

- An XML-RPC server capable of handling requests from coupled models is set up to act as the coupling broker.
- The coupled models set up a link with the coupling broker through XML-RPC
- When a coupled model requires information from another model, it makes a request for that information to the coupling broker.
- Depending on the coupling set-up, the coupling broker forwards this request to the model after modifying it, if necessary, to a format that is understandable by that model.
- The model to which the request is made, processes this information and provides a response to the coupling broker.
- The coupling broker forwards this response to the requesting model, once more adapting it to the appropriate format as necessary.

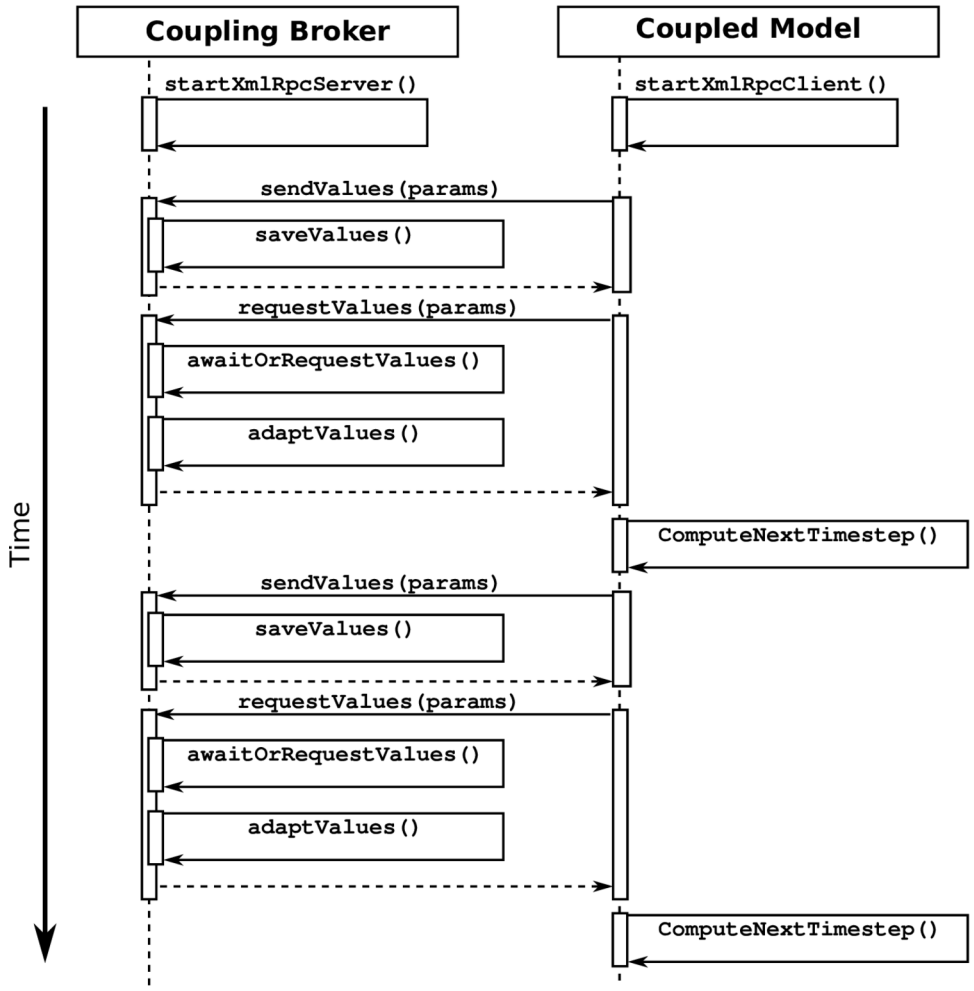

Figure 3.2: Coupling broker concept

- The requesting model makes use of this information.

This coupling mechanism is quite simple in its design and is capable of fulfilling the goals that were set out in the previous section. However, the approaches for representing and adapting information need to be explained in more details and this is done in the following sections.

3.3 Representing information

As has already been mentioned, in order to couple hydroinformatic models, it is beneficial to have a representation of information that is easily adaptable and exists

independent from any of the coupled models. Information units that have these properties can be readily transferred to any coupled model and if required, they can be adapted to the format desired by the coupled model. Such adaptations may include mapping between different spatial discretisations, interpolation, unit conversions, converting between different physical state variables, etc. However, to be able to do so, the information unit should have detailed knowledge about the following:

- dimensions
- units
- values
- coordinate systems
- geometry
- topology
- metadata

Tensor objects are information units that mainly represent the physical state variables and have the exact properties described above. They are a full representation of information needed for coupling and they are easily adaptable. The design of the tensor objects in the current work is based on the concepts proposed by Molkenthin (2000), Molkenthin et al. (2009a,b) and inspired from the OpenMI standard (OpenMI, 2012d, The OpenMI Association Technical Committee, 2010a). The constituents that make up the tensor objects are described in more details in the following sections.

3.3.1 Dimensions

Description: Dimensions, not to be confused with the dimensionality of a coordinate system, are a concept that comes from the field of dimensional analysis in physics (Bridgman, 1922, Buckingham, 1914), which provides a way to quantitatively describe a physical quantity (White, 2003). Dimensional analysis is based on the concept of seven base physical quantities (Gibbings, 2011) based on the SI International System of Units (*Bureau International des Poids et Mesures*, 2006). These are:

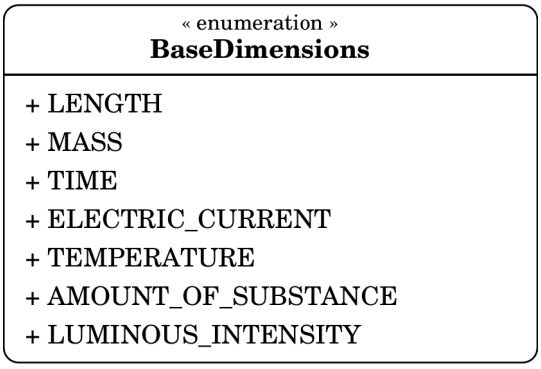

Figure 3.3: `BaseDimensions` enumeration

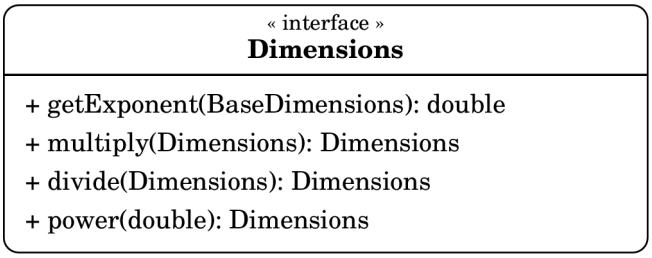

Figure 3.4: `Dimensions` interface

- Length $[\mathrm{L}]$
- Mass $[\mathrm{M}]$
- Time $[\mathrm{T}]$
- Electric current $[\mathrm{A}]$
- Temperature $[\theta]$
- Luminous intensity $[\mathrm{C}]$
- Amount of substance $[\mathrm{n}]$

Any physical quantity can be qualitatively expressed in terms of the exponents of these seven base quantities, e.g. acceleration, force and work can be represented as $[\mathrm{LT}^{-2}]$, $[\mathrm{MLT}^{-2}]$ and $[\mathrm{ML}^{2}\mathrm{T}^{-2}]$ respectively.

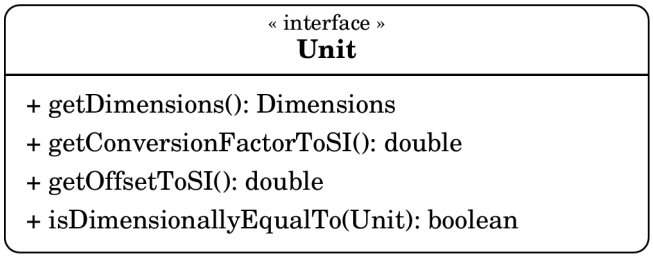

Figure 3.5: `Unit` interface

Interface details: Figures 3.4 and 3.3 respectively show the UML representation of the `BaseDimensions` enumeration for the seven base physical quantities, and the `Dimensions` interface for a concrete instance of dimensions, e.g. $[\mathrm{ML}^2\mathrm{T}^{-2}]$. The `Dimensions` interface provides the following functionality:

- access the exponent for individual base quantities through the `getExponent(BaseDimensions)` method
- compute the `Dimensions` object that is the result of multiplying or dividing a `Dimensions` object with another through the `multiply(Dimensions)` and `divide(Dimensions)` methods respectively, e.g. multiplying the dimensions of force $[\mathrm{MLT}^{-2}]$ with those of displacement $[\mathrm{L}]$ results in the dimensions of work $[\mathrm{ML}^2\mathrm{T}^{-2}]$

3.3.2 Units

Description: Units are, quite simply, the units of a physical quantity, e.g. metres, miles and parsecs are units of length. Units are composed of dimensions and the conversion factors necessary for converting a physical value from those units to the equivalent SI units. Similar to the OpenMI standard, the conversion is done using the the formula:

$$\text{magnitude}_{\text{SI}} = \text{magnitude} \times \text{factor} + \text{offset}$$

where 'magnitude' and '$\text{magnitude}_{\text{SI}}$' are the magnitude of the physical value in the original and the equivalent SI units respectively and 'factor' and 'offset' are the conversion factor and the offset of this unit to the equivalent SI units respectively. As an example, to convert a length from inches to the equivalent SI unit metre, we need to set factor to 0.0254 and offset to 0.0.

Interface details: Figure 3.5 shows the UML representation of the `Unit` interface which provides the following functionality:

- access the `Dimensions` of the `Unit` through the `getDimensions` method
- access to the conversion factor and offset respectively of the `Unit` to the equivalent SI unit through the `getConversionFactorToSI` and `getOffsetToSI` methods respectively
- check the equivalence to another `Unit` through the `isDimensionallyEqualTo(Unit)` method, e.g. Dyne and Newton are both units of force and are dimensionally equal since both have the dimensions $[\mathrm{MLT}^{-2}]$

3.3.3 Quantities and values

Description: A value is an individual measurement of a physical quantity. Values have a coordinate system (see section 3.3.4) and units, and may be composed of one or more components (called quantities in the current work) according to their rank. Figure 3.6 shows a UML representation of this relationship between values and quantities. The different types of values based on their rank are:

- scalar values having a rank of 0, e.g. temperature
- vector values having a rank of 1, e.g. velocity
- matrix values having a rank of 2, e.g. permeability of soil

  ⋮

- tensor values having a rank higher than 2

Interface details: The `Quantity` interface is simple in its composition and consists of the magnitude of the `Quantity` and its `Unit`s (see figure 3.7). Despite this simplicity, the interface provides some very useful functionality to:

- retrieve the magnitude of the `Quantity` in its own units or another equivalent unit through the `getQuantity` and `setQuantity(Unit)` methods respectively
- compare it to another `Quantity` regardless of its `Unit`s but having the same `Dimensions` through the `compareTo(Quantity)` method

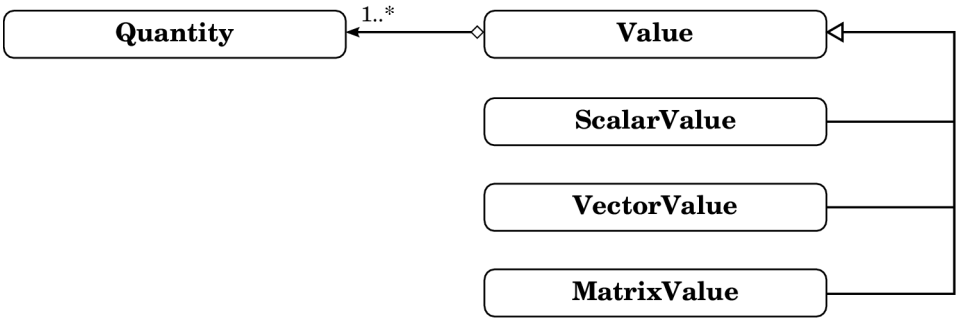

Figure 3.6: `Quantity` and `Value` relationships

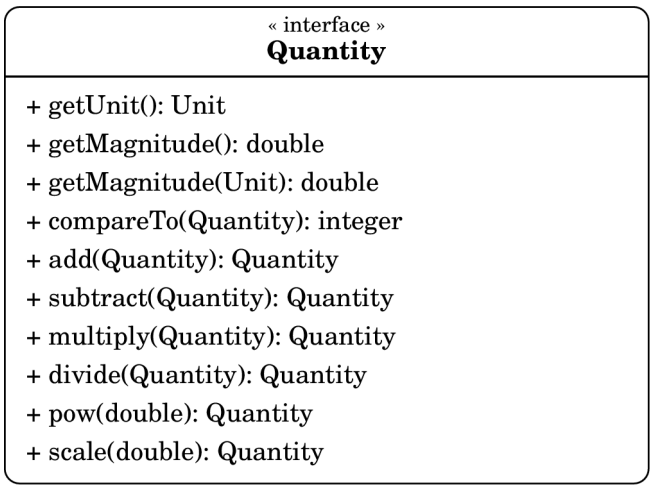

Figure 3.7: `Quantity` interface

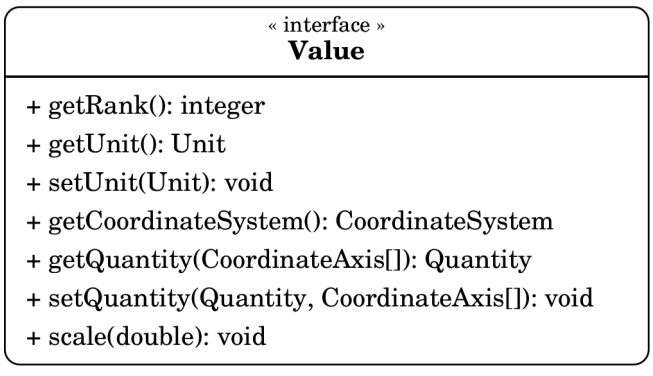

Figure 3.8: `Value` interface

- add another `Quantity` to itself, independent of the other `Quantity`'s `Unit`s, through the `add(Quantity)` method
- subtract another `Quantity` from itself, independent of the other `Quantity`'s `Unit`s, through the `subtract(Quantity)` method
- multiply another `Quantity` to itself through the `multiply(Quantity)` method
- divide another `Quantity` by itself through the `divide(Quantity)` method
- raise itself to a power through the `pow(double)` method
- scale itself by a factor through the `scale(double)` method

A `Value` (see figure 3.8) has a `CoordinateSystem` and is composed of one or more `Quantity` components. Specialised `Value`s of rank zero, one and two are provided by the `ScalarValue`, `VectorValue` and the `MatrixValue` classes. The functionality that the `Value` interface provides includes:

- access the rank, units and coordinate system through the `getRank`, `getUnit` and the `getCoordinateSystem` method respectively
- convert between `Unit`s with the same `Dimensions`, with the corresponding update in the magnitudes in the new `Unit`, through the `setUnit(Unit)` method
- access individual `Quantity` components through the `getQuantity(CoordinateAxis[])` method
- replace individual `Quantity` components through the `setQuantity(Quantity, CoordinateAxis)` method
- scale itself by factor through the `scale(double)` method

3.3.4 Coordinate systems

Description: A coordinate system is a 'system for specifying points using coordinates measured in some specified way' (Weisstein, 2003c) and consists of one or more coordinate axes. The origin of the coordinate system is either some arbitrary point or a specific geodetic datum. A point within a coordinate system can be described by a position vector (see section 3.3.5) to that point. Some examples of coordinate systems are the Cartesian coordinate system, cylindrical coordinate system, geographic coordinate reference systems such as UTM, etc.

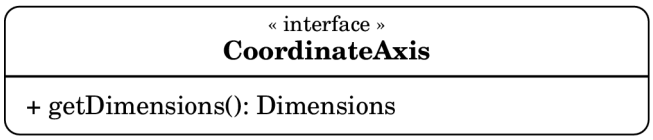

Figure 3.9: `CoordinateAxis` interface

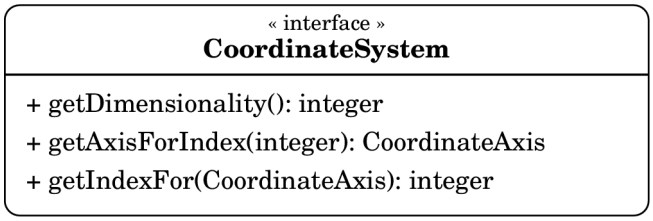

Figure 3.10: `CoordinateSystem` interface

Interface details: The `CoordinateAxis` interface (see figure 3.9) provides a representation for a single coordinate axis. The solitary `getDimensions` method of this interface provides access to the `Dimensions` for valid `Quantity` components of `Value`s and `PositionVector`s along this axis.

The `CoordinateSystem` interface (see figure 3.10) is composed of one or more `CoordinateAxis` instances and provides the following functionality:

- access its dimensionality, i.e. count of its coordinate axes through the `getDimensionality` method
- access the `CoordinateAxis` having particular index through the `getAxisForIndex(integer)` method
- access the index of a particular `CoordinateAxis` through the `getIndexFor(CoordinateAxis)` methods

3.3.5 Position vectors

Description: A position vector to a point is a vector connecting that point to the origin of the coordinate system. Position vectors are described by their components along each of the axes of the coordinate system, or in other words, its coordinates. Since position vectors determine the position of points within a coordinate system, they are an important means of representing the geometry within tensor objects.

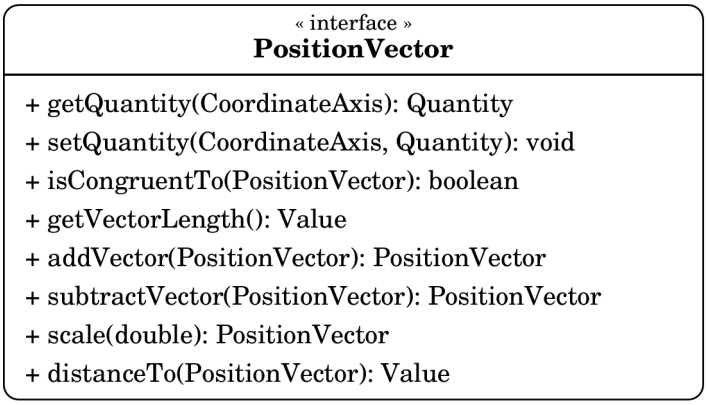

Figure 3.11: `PositionVector` interface

Interface details: The `PositionVector` (see figure 3.11) is composed of a `CoordinateSystem` and one or more `Quantity` components representing its coordinates along each `CoordinateAxis`. Figure 3.11 shows a UML representation of the `PositionVector` interface. The interface provides the following functionality:

- access individual `Quantity` components (coordinates) of the `PositionVector` through the `getQuantity(CoordinateAxis)` method
- replace individual `Quantity` components of the `PositionVector` through the `setQuantity(CoordinateAxis, Quantity)` method
- check the congruency with another `PositionVector` through the `isCongruentTo(PositionVector)` method
- get the magnitude or the l^2-norm (Weisstein, 2003d) of the `PositionVector` through the `getVectorLength` method
- get the result of addition or subtraction of another vector to itself through the `addVector(PositionVector)` and `subtractVector(PositionVector)` methods respectively
- scale itself by the given scaling factor through the `scaleVector(double)` method
- calculate the distance between the points described by two position vectors using the `distanceTo(PositionVector)` methods

3.3.6 Polytopes

Description: A polytope is an arbitrary geometric object that exists in a coordinate system of arbitrary dimensionality (Weisstein, 2003a, Wikipedia, 2015j). The order of a polytope can be described in terms of their dimensionality in the coordinate system, e.g.:

- a 0-polytope is a point
- a 1-polytope is a line or a polyline
- a 2-polytope is a polygon
- a 3-polytope is a polyhedron

  ⋮

- an n-polytope

Within an n-dimensional coordinate system, polytopes of order 1-polytope to n-polytope can possibly exist. An n-polytope can be thought to be composed of an arbitrary number of $(n-1)$-polytopes. For the sake of completeness, the polytope concept includes the concept of $null$-polytopes, which are the constituents of 0-polytopes (points).

Polytopes provide the means for describing the topological relationships within tensor objects. Each n-polytope is composed of a set of lower order polytopes called subpolytopes. These are usually, but don't necessarily have to be, $(n-1)$-polytopes, e.g. a polygon can be described either as a set of its edges or its vertices. Polytopes can also contain information about any higher order polytopes, called superpolytopes, of which they are constituents. These are usually the $(n+1)$-polytopes but may also have a higher order, e.g. depending upon the design of a polygon, its vertices may contain information about either the edges of the polygon or the polygon itself. Therefore, a tree-like structure representing all the subpolytope-superpolytope relationships is formed and it is possible to walk up or down this tree to find related polytopes of any order for any particular polygon.

Interface details: Figure 3.12 shows the UML representation of the `Polytope-Order` interface that contains information about the implementation-dependent polytope-order hierarchy. It provides the following functionality:

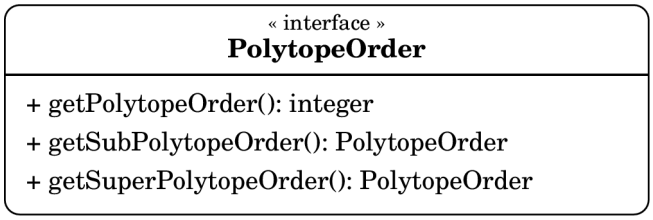

Figure 3.12: The `PolytopeOrder` interface

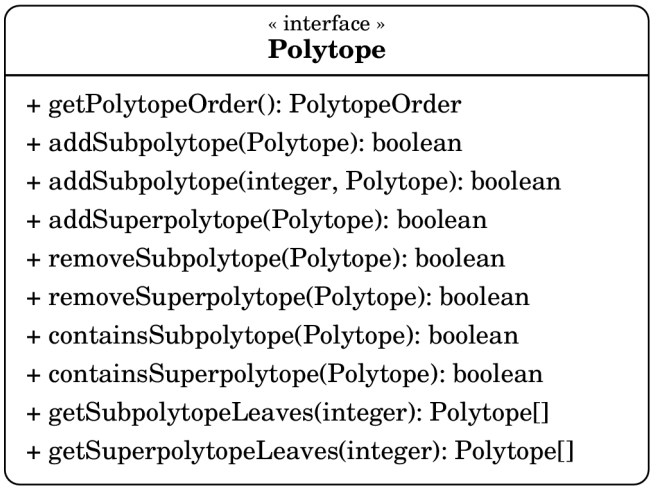

Figure 3.13: The `Polytope` interface

- access the integral value of the polytope-order through the `getPolytope-Order` method
- access the implementation-dependent subpolytope or superpolytope order using the `getSubpolytopeOrder` and the `getSuperpolytopeOrder` methods respectively

A `Polytope` (see figure 3.13) is composed from its `PolytopeOrder` and a set of sub`Polytope`s. Additionally, it contains information about its super`Polytope`s. The interface provides functionality to:

- access the `getPolytopeOrder` through the `getPolytopeOrder` method
- append a sub`Polytope`, add a sub`Polytope` at a particular position or append a super`Polytope` through the `addSubpolytope(Polytope)`, `addSubpoly-tope(integer, Polytope)` and `addSuperpolytope(Polytope)` methods

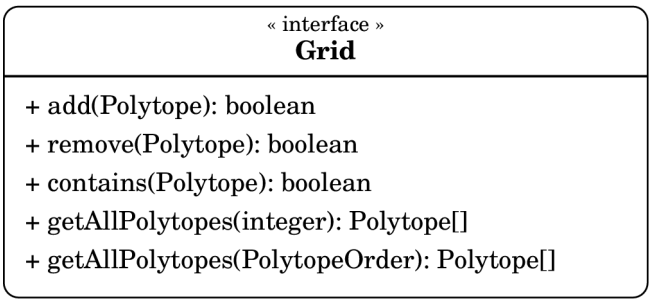

Figure 3.14: The `Grid` interface

respectively

- remove a sub`Polytope` or super`Polytope` through the `removeSubpolytope(Polytope)` and `removeSuperpolytope(Polytope)` methods respectively
- check whether or not a `Polytope` is its sub`Polytope` or super`Polytope` through the `containsSubpolytope(Polytope)` and the `containsSuperpolytope(Polytope)` methods respectively
- retrieve sub`Polytope`s or super`Polytope`s at an arbitrary depth in the subpolytope-superpolytope hierarchical tree through the `getSubpolytopeLeaves(integer)` and `getSuperpolytopeLeaves(integer)` methods respectively

3.3.7 Grids

Description: A grid is a discretisation of a model domain, e.g. a computational grid for the solution of partial differential equations. Computational grids basically consist of a distribution of computational nodes over the whole domain that are interconnected in some way (Carey, 1997) or a topographic grid within a digital elevation model. It is possible to have grids that are structured or unstructured, regular or irregular, Cartesian or non-Cartesian and so on (Thompson et al., 2010). A grid represents both the geometric and topological relationships among its elements. A grid is also capable of representing geometric and topological relationships for physical state variables whose coordinate systems aren't spatial in nature, e.g. points in a stress-strain relationship. A grid is composed of polytopes, where:

- 0-polytopes of any arbitrary grid element represent its geometry, and

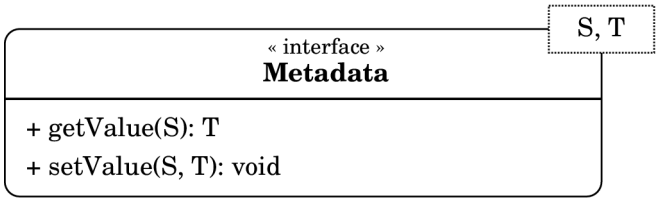

Figure 3.15: The `Metadata` interface

- the subpolytope and superpolytope relationships of a grid element represent the topological relationships

Interface details: Figure 3.14 shows the UML representation of the `Grid` interface. A `Grid` is composed of a collection of `Polytope`s and provides functionality to:

- add or remove `Polytope`s to the `Grid` through the `add(Polytope)` and `remove(Polytope)` methods respectively
- test the presence of a `Polytope` in the `Grid` through the `contains(Polytope)` method
- retrieve all `Polytope`s of a particular `PolytopeOrder` in the `Grid` through the `getAllPolytopes(PolytopeOrder)` or the `getAllPolytopes(integer)` methods

3.3.8 Metadata

Description: Metadata, which can be defined as data about data, has already been described in details in the section 2.1.1. That description is therefore not repeated here. For the current work a general approach to store metadata in the form of key-value pairs is used instead of choosing a single metadata standard such as Dublin Core or ISO 19115.

Interface details: Figure 3.15 shows a UML representation of the `Metadata` interface. The interface is very simple in its implementation and is simply a mapping of metadata keys to the corresponding metadata values. It provides a way to access and set metadata values through the `getValue(S)` and `setValue(S, T)` method respectively.

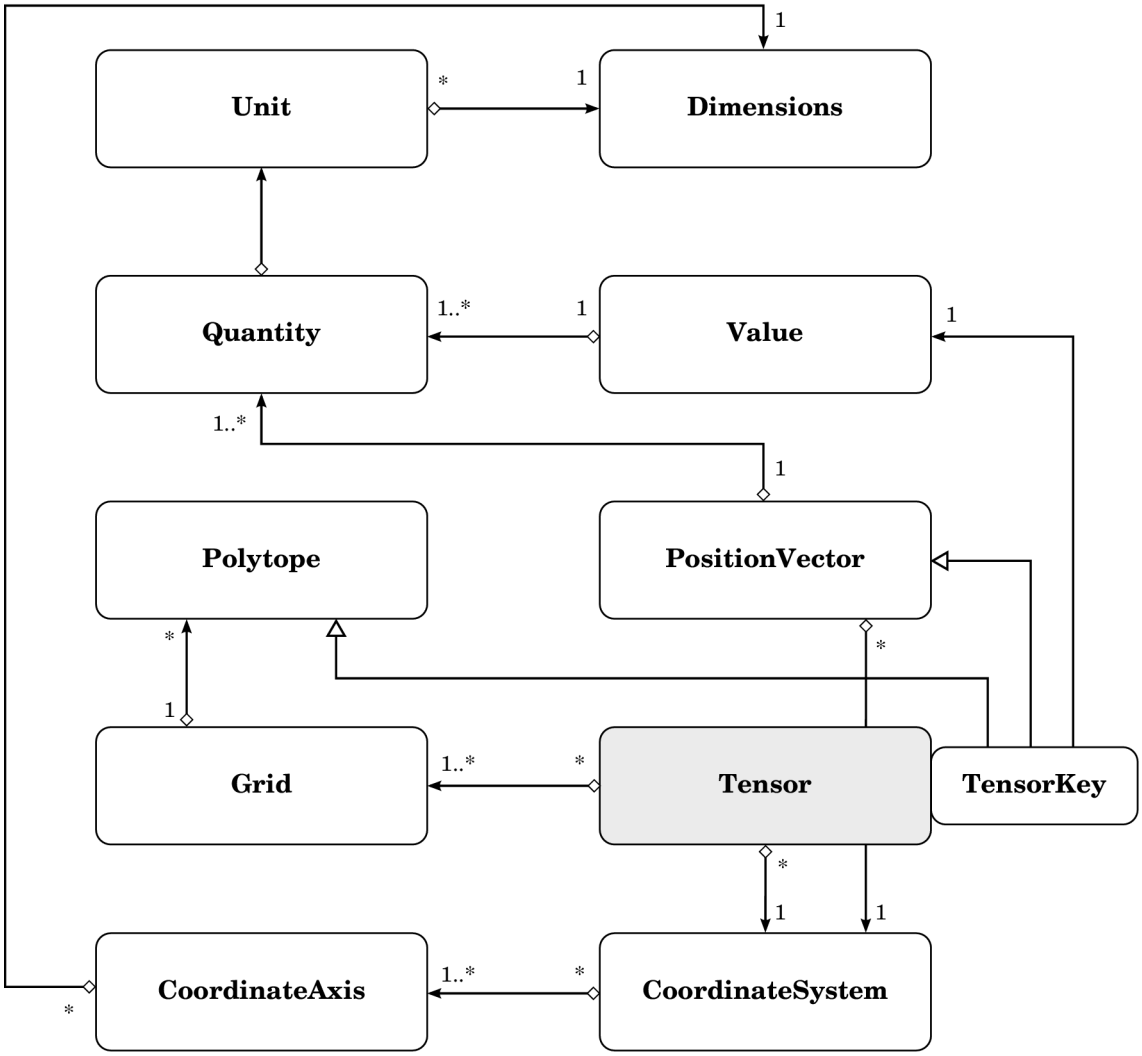

Figure 3.16: `Tensor` components

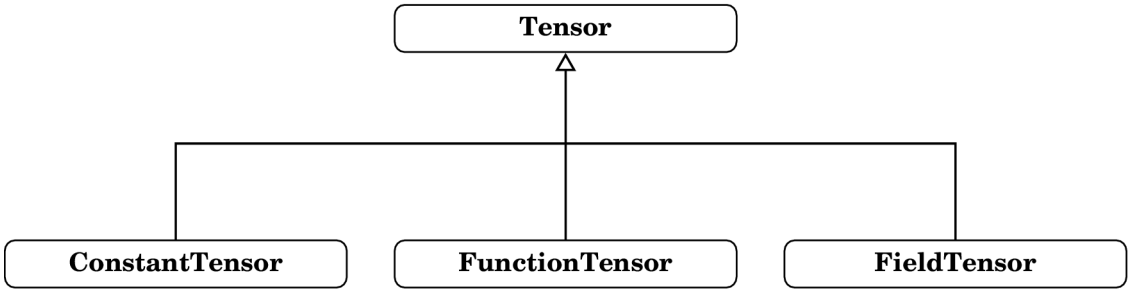

Figure 3.17: Specialised `Tensor` objects

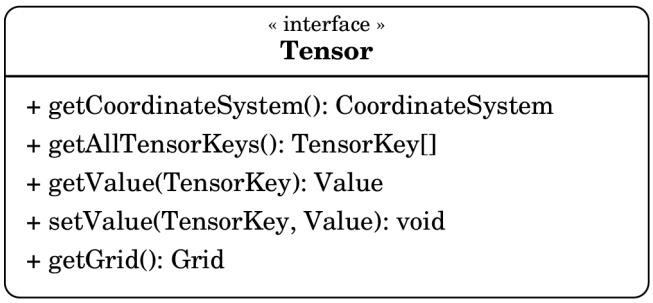

Figure 3.18: The `Tensor` interface

3.3.9 Tensor object

Description: The tensor object, as mentioned earlier, is a comprehensive representation of information. Figure 3.16 shows the UML representation of the composition of a tensor object. As implied by this figure, the tensor object contains the following information:

- **dimensions and units** of each of the tensor object's components
- **values** in the form of discrete measurements within a physical state variable
- **coordinate system** of the tensor object and of the individual values. As shown in figure 3.17, based on their dimensionality, tensor objects may be:
  - **constant** tensor objects that are independent of coordinate systems. They might mathematically be represented as $z = C$.
  - **function** tensor objects with one independent coordinate axis. Mathematically, they may be represented as $z = f(x)$, e.g. velocity time series.
  - **field** tensor objects with two independent coordinate axes. Mathematically, they may be represented as: $z = f(x, y)$, e.g. a digital elevation model or a stress-strain relationship.

    $\vdots$

    and so on.
- **geometry** as stored in the constituent grid of the tensor object
- **topology** as stored in the constituent grid of the tensor object

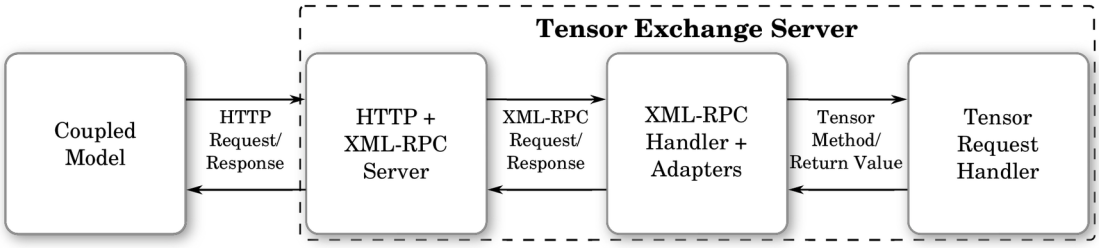

Figure 3.19: Schematic diagram of the TES in action

Interface details: Figure 3.18 shows a UML representation of the `Tensor` interface. Here, the `TensorKey` combines the geometric and topological information since it is a specialisation of both a `PositionVector` and a `Polytope` (see figure 3.16). The `Tensor` object is therefore a mapping of `TensorKey`s to `Value`s. The functionality provided by the `Tensor` interface includes:

- access the coordinate system of the `Tensor` object through the `getCoordinateSystem` method
- get a list of all the `TensorKey`s through the `getAllTensorKeys` method
- get or set the `Value` located a particular position through the `getValue(TensorKey)` and `setValue(TensorKey, Value)` methods respectively
- access the grid of the `Tensor` object through the `getGrid` method

3.4 Coupling broker

The final piece of technology that is necessary for the coupling of models is the coupling broker as the Tensor Exchange Server (TES) that acts as the broker for communication among the models, information exchange among coupled models and the adaptation of this information to the requirements of the coupled models. The coupling broker performs the following functions:

- communicate with the individual coupled models over a network connection
- decode information requests/responses in the form of remote procedure calls (RPC) from the coupled models to:
  - extract information from the data stream

  - interpret the information request from the RPC for forwarding it to the appropriate model

- convert information to and from `Tensor` objects to the native representation that the coupled model understands if necessary
- adapt `Tensor` objects where and when appropriate, e.g. perform mapping between coordinate systems, transform one physical state variable to another through the application of mathematical operations such as multiplication, division, integration, etc.
- encode the `Tensor` objects in the form of RPC responses for the coupled models
- perform simulation control in coupled set-ups by either waiting until the information requested by one model is provided by another coupled model or actively forwarding this request to such a model.

As implied by the list above, the coupling broker is responsible for brokering the communication between the models. It is responsible for handling the information provided by the coupled models and adapting this information to the requirements of the coupled models when requested by them.

A schematic representation of the entire sequence of events that are involved in the coupling of models through the coupling broker is shown in figure 3.19.

3.4.1 Handling information

When coupling models using the coupling broker, there might be cases where the model that needs to be coupled doesn't implement the necessary interfaces for working with tensor objects described in section 3.3. In such a case, an additional step involving the conversion of information the native representation of the coupled model to the tensor objects and vice versa is necessary. For this purpose a special handler is required, for example the `CouplingHandler` shown in listing 3.1. Here, the methods `getValueAt(Double, Double)` (lines 16 to 21) and `setValueAt(Double x, Double v)` (lines 27 to 34) have been designed to work with models that don't work with tensor objects. However, the handler works internally by storing this information in tensor objects, as demonstrated by the method `getValueAt(Tensor t, PositionVector p)` (lines 23 to 25), which is finally responsible to handle the request from the `getValueAt(Double, Double)` method (delegation to this method can be seen in line 20). Models that implement the `Tensor` interfaces, are able to

use the `getValueAt(Tensor t, PositionVector p)` method directly, without the need for an additional handler.

Listing 3.1: Handler for coupling requests

```
1  package hydroinformatics.xmlrpc;
2
3  import hydroinformatics.information.*;
4  import hydroinformatics.quantities.*;
5  import hydroinformatics.util.*;
6
7  public class CouplingHandler {
8
9     private SIUnit METRE;
10    private BinaryOperator<Tensor, PositionVector, Double>
11       interpolator;
12
13    private final Tensor tensor0;
14    private final Tensor tensor1;
15
16    public Double getValueAt(Double x, Double y) {
17       PositionVector pos = new PositionVector2D();
18       pos.setQuantity(0, new ScalarValue(METRE, x));
19       pos.setQuantity(1, new ScalarValue(METRE, y));
20       return getValueAt(tensor, pos);
21    }
22
23    public Double getValueAt(Tensor t, PositionVector p) {
24       return interpolator.operate(t, p);
25    }
26
27    public void setValueAt(Double[] xs, Double[] vals) {
28       for(int i=0; i<x.length; i++) {
29          PositionVector pos = new PositionVector1D();
30          pos.setQuantity(new ScalarValue(METRE, xs[i]));
31          Value val = new ScalarValue(METRE, vals[i]);
32          tensor1.setValue(pos, val);
33       }
34    }
35 }
```

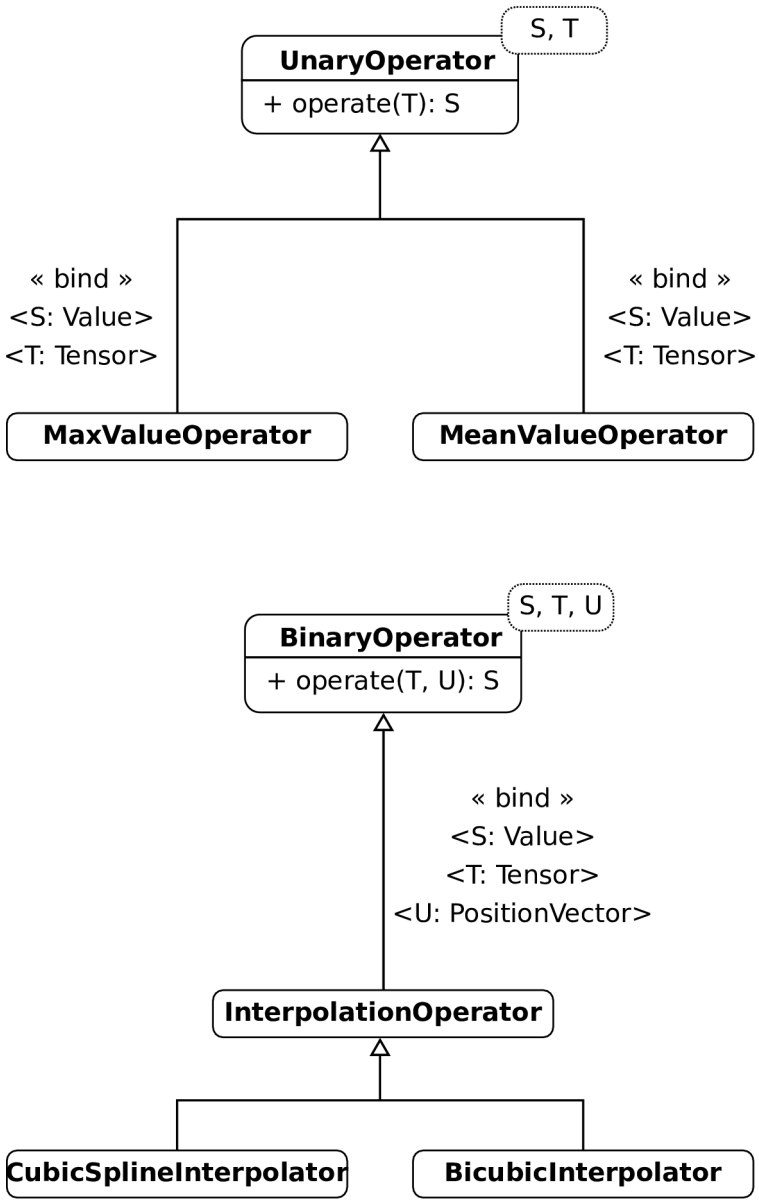

Figure 3.20: Operators for adapting `Tensor` objects

3.4.2 Adapting information

The structure of tensor objects was discussed in section 3.3. Tensor objects are able to completely represent information including units, values, coordinate systems, geometry and topology. Such a representation is useful for coupling models because it enables the adaptation of information according to the requirements of the coupled model. In order to achieve such an adaptation by the coupling broker, the concept of `Operator`s is introduced. An `Operator` takes an input, performs an operation using that input and gives back the result of the operation. Figure 3.20 shows a UML representation of two types of operators:

- **Unary operator**: The `UnaryOperator` accepts a single input parameter to operate upon and returns the result of the operation. A unary operator that takes a tensor object as an input may be used to compute properties such as the mean, minimum or maximum values of that tensor object.
- **Binary operator**: The `BinaryOperator` is similar to the `UnaryOperator` in every aspect except that it operates on two input parameters, e.g. a binary operator may:
  - operate on a tensor object and a point (position vector) to calculate the interpolated value at that point
  - operate on a tensor object and a grid to map values from the tensor object to the given grid

If it is required, it is possible to implement higher order operators using the same principle. However, it is simpler to limit oneself to the operators described above and to simulate higher order operators by using a collection of parameters as a single input parameter for either the unary or binary operator.

3.5 Conclusions

The concept of using tensor objects for representing information and operators for adapting this information provides way to separate the representation of the data from the adaptation functions. The advantages of this approach may be summarised as follows:

- **Simplicity**: The structure of tensor objects can be kept simple because they don't have to anticipate the requirements of the coupled models.

- **Extensibility**: It is possible to provide newer functions for adapting information in the future as requirements change because of changes in the coupling set-up.
- **Flexibility**: The tensor objects and operators can be developed and maintained independent of each other.
- **Reusability**: An operator may operate on multiple tensor objects without having to reimplement it. Similarly, separate operators may be used to operate on the same tensor object according to requirements.

Chapter 4

Model coupling prototype implementation

Based on the concept described in the previous chapter, a prototype for a framework for coupling hydroinformatic models has been developed using the Java programming languages. The implementation details of this prototype are described in the following sections.

4.1 Tensor objects

As described in section 3.3, the current work makes use of tensor objects to represent information. Constituents of tensor objects that are related to each other have been grouped into the following four packages:

- quantities
- geometry
- topology
- information

These packages are described in more details in the following sections:

4.1.1 Quantities

The `quantities` package (see figure 4.1) consists of all classes that are necessary to describe quantities and values (see section 3.3.3). This includes:

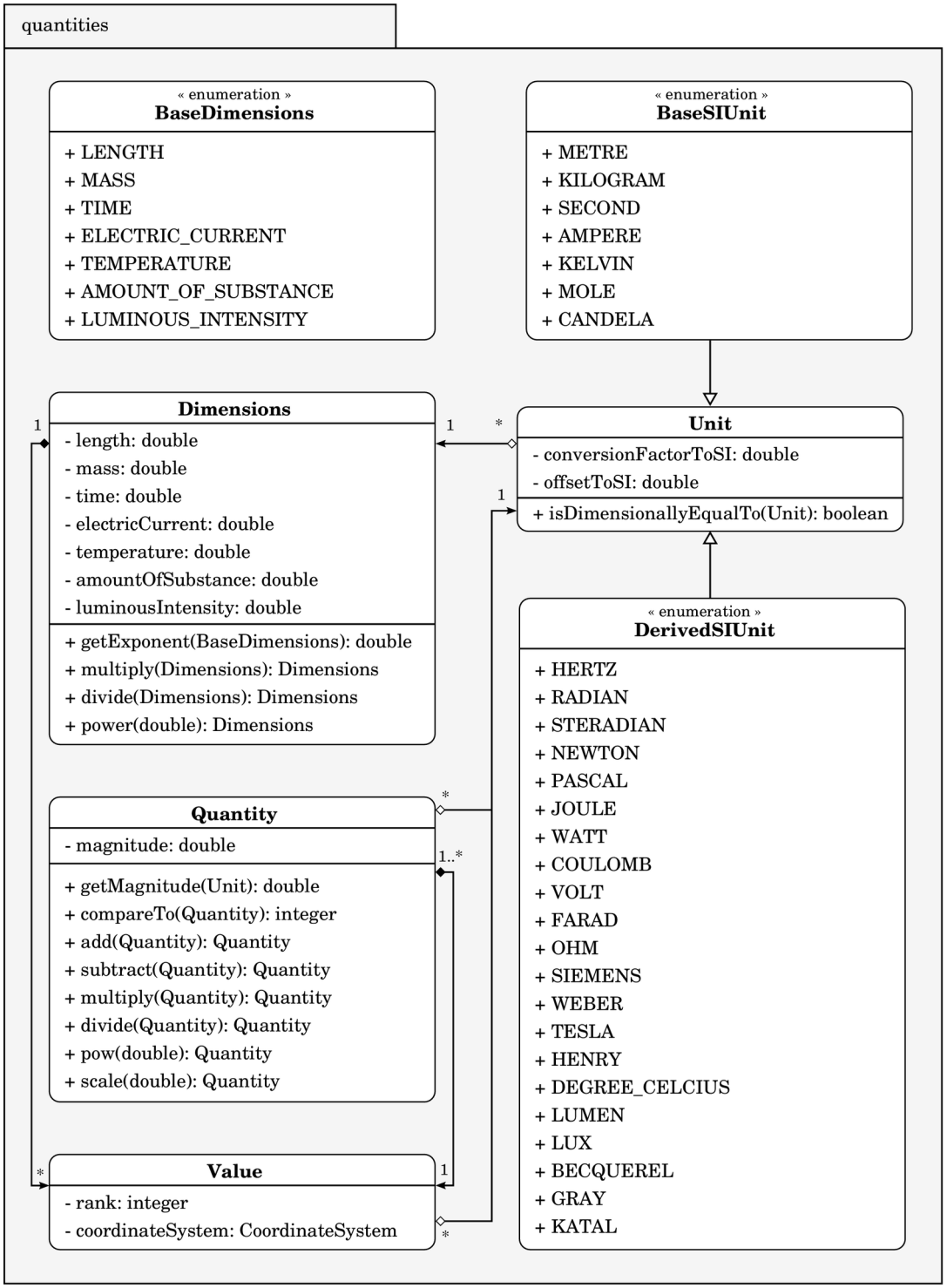

Figure 4.1: The `quantities` package

- The `BaseDimension` enumeration for the seven base physical quantities (see section 3.3.1).
- The `Dimensions` class for representing the dimensions of a value. It does this by storing the exponents of the seven `BaseDimension`s. Section 3.3.1 provides a detailed discussion of the functionality provided by the `Dimensions` class.
- The `Unit` class to represent the units of a value. It is composed of a `Dimensions` object and two numbers representing the conversion factor and offset of the unit to the equivalent SI unit. Section 3.3.2 provides a detailed discussion of the functionality provided by the `Unit` class.
- `BaseSIUnit` and `DerivedSIUnit` utility enumerations that provide implementations of the base and derived SI units respectively as defined by the *Bureau International des Poids et Mesures* (2006).
- The `Quantity` class, which is useful for describing values and position vectors, is composed of a `Unit` and a number representing the magnitude of the quantity. Section 3.3.3 provides a detailed discussion of the functionality provided by the `Quantity` class.
- The `Value` class for representing a generic physical value. It consists of:
  - the integral value of the rank of the value
  - the coordinate system of the value
  - the dimensions for the valid quantity components of the value
  - one or more quantity components of the value

  Section 3.3.3 provides a detailed discussion about the functionality provided by the `Value` class.

4.1.2 Geometry

The `geometry` package (see figure 4.2) contains classes necessary to describe the geometeric properties of tensor objects. These include:

- The `CoordinateAxis` class, which is very simple in its composition and consists only of a `Dimensions` object for valid `Quantity` components along this axis for `Value`s and `PositionVector`s. Section 3.3.4 provides a detailed discussion about the functionality provided by the `CoordinateAxis` class.

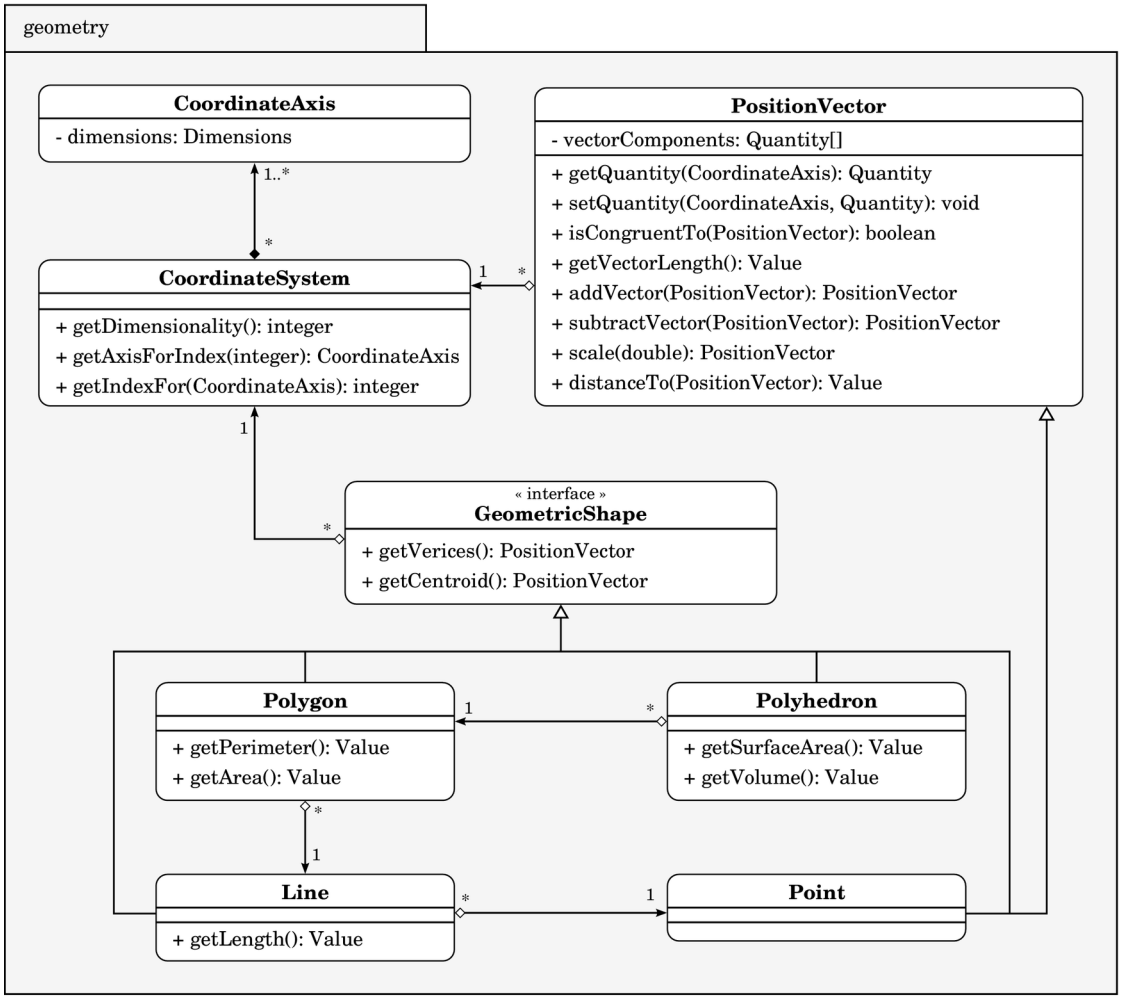

Figure 4.2: The `geometry` package

- The `CoordinateSystem` class for representing coordinate systems. It is composed of one or more `CoordinateAxis` instances. Section 3.3.4 provides a detailed discussion about the functionality provided by the `CoordinateAxis` class.
- The `PositionVector` class for representing position vectors. It has a `CoordinateSystem` and is composed of one or more `Quantity` components for each of the axes of its coordinate system. Section 3.3.5 provides a detailed discussion about the functionality provided by the `CoordinateAxis` class.
- The `GeometricShape` interface that provides access to common geometric

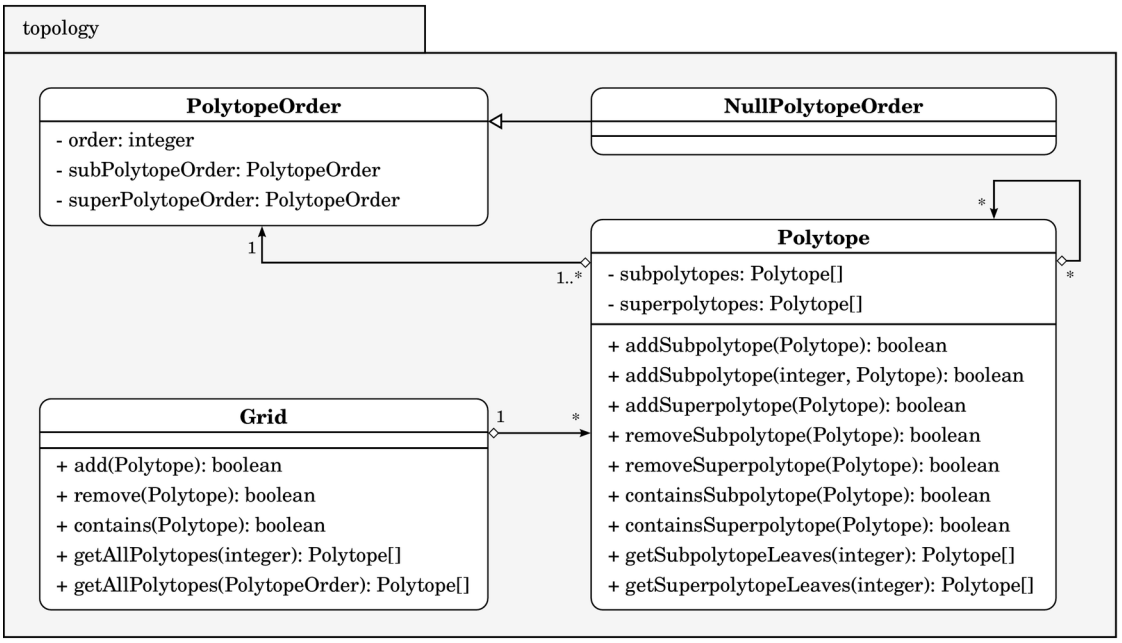

Figure 4.3: the `topology` package

properties of geometric shapes: their vertices and centroid through the `getVertices` and `getCentroid` methods respectively.

- Concrete implementations of zero-, one-, two- and three-dimensional `GeometricShape`s in a spatial coordinate-system, i.e. the `Point`, `Line`, `Polygon` and `Polyhedron` classes respectively. These classes provide additional functionality relevant to the shape that they represent, which is:
  - the length of the `Line` through the `getLength` method
  - the perimeter and area of the `Polygon` through the `getPerimeter` and `getArea` methods respectively
  - the surface area and volume of the `Polyhedron` through the `getSurfaceArea` and `getVolume` methods respectively

4.1.3 Topology

The `topology` package (see figure 4.3) contains classes necessary to describe the geometeric properties of tensor objects. These include:

- The `PolytopeOrder` class for representing polytope orders. It is composed of the integral value of the polytope order and sub- and super-polytope orders of that polytope orders. Section 3.3.6 provides a detailed discussion about polytope orders and the functionality provided by the `PolytopeOrder` class.

- The `NullPolytopeOrder` utility class in order to represent subpolytopes of 0-polytopes or the superpolytopes of n-polytopes in an n-dimensional coordinate system.

- The `Polytope` class for representing polytopes. It is composed of a `PolytopeOrder` of the polytope and lists of its subpolytopes and superpolytopes. Section 3.3.6 provides a detailed discussion about the functionality provided by the `Polytope` class.

  The `Grid` class for representing a computational grid. It is composed of a collection of all the `Polytope`s that form the grid. Section 3.3.7 provides a detailed discussion about the functionality provided by the `Grid` class.

4.1.4 Information

The `information` package (see figure 4.4) provides the final remaining tensor components and in combination with the components described above, it provides an implementation of the autonomous tensor objects. The package consists of:

- The `Metadata` class for describing metadata of a tensor object. It consists of a mapping of metadata keys to the metadata values. Section 3.3.8 provides a detailed discussion about the functionality provided by the `Metadata` class.

- The `Tensor` class for representing the tensor objects. It is a mapping of `TensorKey`s to `Value`s that are part of the tensor object. Section 3.3.9 provides a detailed discussion about the tensor keys, which represent the geometric and topological relationships of the tensor object, and the functionality provided by the `Tensor` class.

- The `TensorSet` class which is a collection of `Tensor` objects that are related in some way to each other, e.g. by sharing a grid.

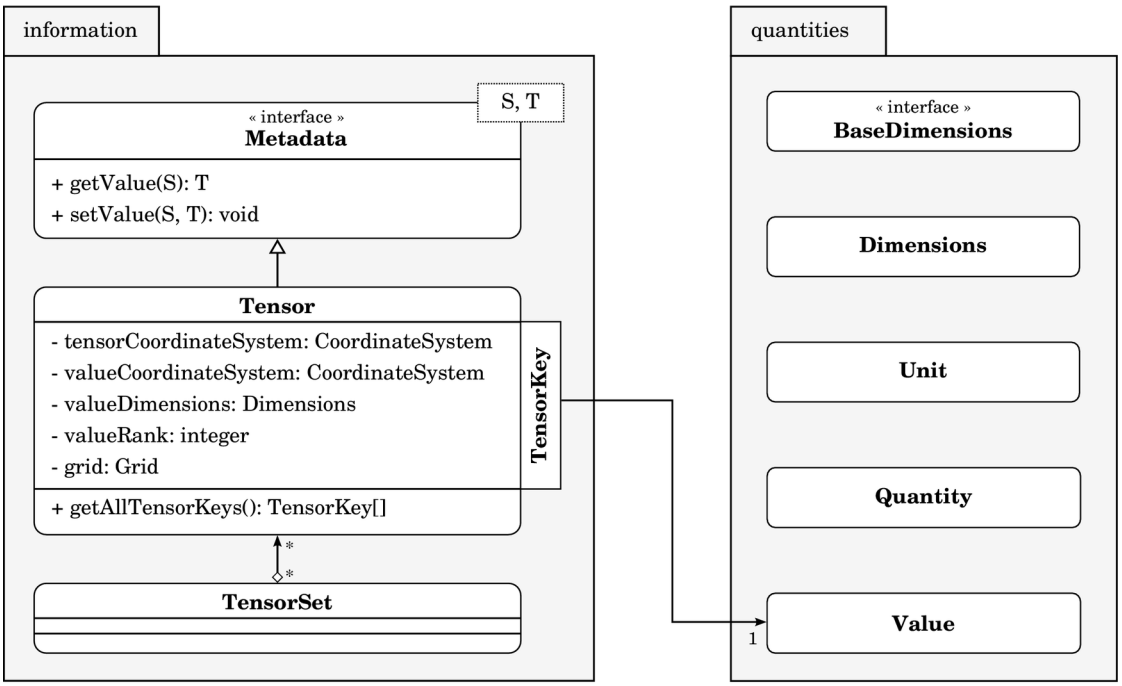

Figure 4.4: UML representation of a quantities package

4.2 Operators

In order to adapt the information from the `Tensor` objects, the prototype provides implementations of some important `Operator`s (see section 3.4.2). These include:

- `UnaryOperator`s using a shape as input for calculating its geometric properties, e.g. `CentroidCalculator`, `PolygonAreaCalculator`, etc.
- `UnaryOperator`s taking a `Tensor` object as an input for outputting files in formats such as comma separated values, VTU, XML, etc. and vice versa.
- `UnaryOperator`s taking a `Tensor` object as an input for converting it to a data stream in the preferred representation of a coupled model and vice versa.
- `BinaryOperator`s that take a `Tensor` object and a `PositionVector` input to calculate the interpolated value of the position described by the position vector, e.g. `LinearInterpolator`, `SplineInterpolator`, etc.
- `BinaryOperator`s that take a `Tensor` object and a `CoordinateSystem` as input for performing coordinate system transformations, e.g. converting time

between the Julian Day (Wikipedia, 2015d) and Unix-time (Wikipedia, 2015n) epochs using `LongToModifiedJulianDateConvertor`.

- `BinaryOperator`s that take two `Tensor` objects as input for mapping values from one tensor object to the grid of the other.

4.3 Coupling broker

The concept of the coupling broker was introduced in section 3.4. The implementation of the coupling broker for the prototype of the model coupling framework developed for this work has been implemented in Java and has been named as the Tensor Exchange Server (TES). It uses the Apache XML-RPC library (Foundation, 2010) that acts as the server for communicating with the coupled models.

Listing 4.1 shows an implementation of an XML-RPC server in Java via the `CouplingServer` class that acts as the starting point for the Tensor Exchange Server application which functions as the coupling broker. Here, in lines 5 – 7 the classes necessary to set up the XML-RPC server are imported from the Apache XML-RPC library. In order to do so, the appropriate jar files from the Apache library need to be added to the classpath of the Java application (Oracle, 2015a). In this example the server is started (line 49) on port 8093 (lines 24 and 30). The rest of the main method is for configuring the settings of the XML-RPC server.

Listing 4.1: XML-RPC server for coupling models

```
1  package hydroinformatics.xmlrpc;
2
3  import hydroinformatics.coupling.*;
4
5  import org.apache.xmlrpc.XmlRpcException;
6  import org.apache.xmlrpc.server.*;
7  import org.apache.xmlrpc.webserver.WebServer;
8
9  /**
10  * Class for starting the Tensor Exchange Server
11  * - initialises the XML-RPC server
12  * - initialises the Tensor set for coupling
13  * - sets the handler for the requests by the
14  *   the coupled models via XML-RPC requests
```

```
15  */
16 public class CouplingServer {
17
18    /*
19     * Initialise the tensor set with tensor objects
20     * that can be accessed throughout the TES
21     */
22    public static final TensorSet TENSOR_SET
23          = new TensorSet();
24    private static final int serverPort = 8093;
25
26    public static void main(String[] args) {
27       try {
28          // Set up and start the XML-RPC server
29          WebServer webServer
30                = new WebServer(serverPort);
31          XmlRpcServer xmlRpcServer
32                = webServer.getXmlRpcServer();
33          PropertyHandlerMapping phm
34                = new PropertyHandlerMapping();
35          /*
36           * Set handler for requests by the
37           * coupled models
38           */
39          phm.addHandler("CouplingHandler",
40                         CouplingHandler.class);
41          xmlRpcServer.setHandlerMapping(phm);
42
43          XmlRpcServerConfigImpl serverConfig
44                = (XmlRpcServerConfigImpl)
45                      xmlRpcServer.getConfig();
46          serverConfig.setEnabledForExceptions(true);
47          serverConfig.setContentLengthOptional(false);
48
49          webServer.start();
50       } catch (XmlRpcException e) {
51          e.printStackTrace();
52       } catch (IOException e) {
53          e.printStackTrace();
54       }
55    }
56
57 }
```

Listing 4.2: Handler for XML-RPC requests

```
1  package hydroinformatics.xmlrpc;
2
3  import hydroinformatics.coupling.*;
4  import hydroinformatics.information.*;
5  import hydroinformatics.util.*;
6
7  public class CouplingHandler {
8
9     private final BinaryOperator
10       <CouplingTensor, Double[], Double>
11          interpolator = new TensorWaterDepthInterpolator();
12    private final CouplingTensor tensor
13          = CouplingServer.TENSOR_SET.WATER_DEPTHS;
14
15    public Double getValueAt(Double x, Double y) {
16       return interpolator.operate(
17             tensor,
18             new Double[] { x, y });
19    }
20 }
```

When the XML-RPC server, i.e. the coupling server receives a request from the XML-RPC client, i.e. a coupled model, it hands over this request to a so-called coupling handler. In listing 4.1, the coupling handler is set on lines 39 to 40. The XML-RPC clients can call any public method of the handler using the `<handler name>.<method name>` format. It is therefore this class that is responsible for handling the requests from the XML-RPC clients. Listing 4.2 shows an example of an XML-RPC handler with a method for interpolating values in a tensor object at a point with the given coordinates. The XML-RPC client can trigger this method by referring to it as the `CouplingHandler.getValueAt` method in its XML-RPC request (also see line 8 of listing 2.1).

Lines 22 to 23 of listing 4.1 also refer to a static tensor set associated with this instance of the coupling broker. The tensor set is the collection of all the tensors that are required in the coupling process. It is put in the same class as the `main` method so that the tensor set is automatically existent as a static object when the coupling broker starts. Being a `public static` field also means that it is accessible

from anywhere within the application. Listing 4.3 shows an example of a tensor set implementation. In this case, only one tensor representing the water depths is shown. The `CouplingTensor`, whose implementation is shown in listing 4.4, is a utility class that implements the `Tensor` interface and also initialises various important components of the tensor object, such as the coordinate system, grid, etc. Depending on the coupling requirements, the implementation of this class may differ or it may also be possible to have more than one such utility class.

Listing 4.3: Example of a tensor set for use in coupling

```
1  package hydroinformatics.coupling;
2
3  public class TensorSet {
4
5     private static SIUnit metre = BaseSIUnit.METRE;
6     private SIUnit fluxUnits;
7     private static Dimension dim;
8
9     static {
10       dim = metre.getDimensions().pow(3);
11       dim = dim.divide(BaseSIUnit.SECOND.getDimensions());
12       fluxUnits = BaseSIUnit.getGenericSIUnitFor(dim);
13    }
14
15    /*
16     * Tensor objects representing the
17     * digital elevation model and fluxes
18     */
19    public static final CouplingTensor WATER_DEPTHS
20          = new CouplingTensor(metre);
21    public static final CouplingTensor FLUXES
22          = new CouplingTensor(fluxUnits);
23
24    // Other tensor objects in the tensor set
25
26 }
```

Listing 4.4: Example of a tensor object for use in coupling

```
1  package hydroinformatics.coupling;
2
3  import hydroinformatics.quantities.*;
4  import hydroinformatics.topology.BasicGrid;
5  import hydroinformatics.information.*;
6  import hydroinformatics.util.*;
7
8  import java.util.*;
9
10 public class CouplingTensor implements
11       Tensor<CouplingNode, ScalarValue, BasicGrid> {
12
13    // Set important properties of tensor object
14    private SIUnit valueUnits = BaseSIUnit.METRE;
15    private CoordinateSystem csTensor
16          = new CouplingCoordinateSystem();
17    private CoordinateSystem csValue = ScalarCoordinateSystem
18          .getInstance(valueUnits.getDimensions());
19    private Grid grid = new CouplingGrid();
20    private MetadataStore metadata = new BasicMetadataStore();
21
22    private Map<double[], Double> elevations = new HashMap<>();
23
24    /*
25     * Methods for accessing or modifying information that
26     * the tensor object contains about the physical state
27     * variable
28     */
29 }
```

In order to process the request of the coupled models (XML-RPC clients), the `CouplingHandler` class also makes use of an operator for processing the information from tensor objects (lines 10, 11 and 16 to 18 of listing 4.2). Listing 14 in the appendix shows an example of an `Operator` for the linear interpolation of values in a tensor object in time and space. Depending on the requirements of the coupled application, it is also possible to have more than one `Operator` implementation. So for the same tensor object, it might be possible to provide both a linear interpolator and a spline interpolator.

To summarise the *modus operandi* of the coupling broker:

- The `CouplingServer` starts an XML-RPC server that acts as the interface between the coupling broker and the coupled models. It listens for XML-RPC requests from the coupled models on a predefined port.
- The `CouplingServer` initialises a tensor set for use in coupling.
- On receiving a request from a coupled model, the XML-RPC server hands over the request to the `CouplingHandler` class. Depending on the implementation, `CouplingHandler` class either waits until the information necessary to fulfil this request is provided by the other model, or requests this information from that model.
- The `CouplingTensor` class implements the `Tensor` interface and also performs some useful initialisations such as setting the coordinate systems, grid, etc. of the tensor object according to the coupling requirements. The exact implementation or count of such utility tensor object classes can be adapted to the coupling requirements.
- The `CouplingHandler` makes use of `Operator` implementations for working with the information in tensor objects. Using different operators, it is possible to adapt the information from tensor objects to different model requirements.

Chapter 5

Applications

This chapter discusses application examples demonstrating the utility of tensor objects in coupling of hydroinformatic models. It begins with a section describing all the models used in the application examples, with the subsequent sections describing the coupled models themselves.

5.1 Simulation models

The applicability of the implemented software prototype for coupling models has been demonstrated by coupling a variety of hydroinformatic models e.g. simulation models, database models, GIS models, etc. The specific models used in the application examples are described in the following sections.

5.1.1 Hyporheic zone metabolism: laboratory model

The set-up for the laboratory model simulating the Hyporheic zone (Hz) consists of a set of percolating microcosms ($20\,\mathrm{ml}$ glass syringes, diameter $2.01\,\mathrm{cm}$, Fortuna Optima, Poulten & Graf, Werheim, Germany; peristaltic pump Ecoline; ISMATEC, Glattbrugg, Switzerland) filled with sediment and placed in a water bath at constant temperature ($15\,^{\circ}\mathrm{C}$) and in darkness (F38-EH; Julabo, Seelbach, Germany). The sediments used had different permeabilities and grain sizes (gravel: 4 to $8\,\mathrm{mm}$, sand: 0.4 to $0.8\,\mathrm{mm}$), but similar porosities (0.4). They were pre-incubated ($3\,\mathrm{d}$, $20\,^{\circ}\mathrm{C}$ in darkness) in a sediment community solution from the experimental catchment Chicken Creek (Gerwin et al., 2009). Different sediment arrangements, both homogeneous and heterogeneous (see figure 5.1), were used for measurements with each

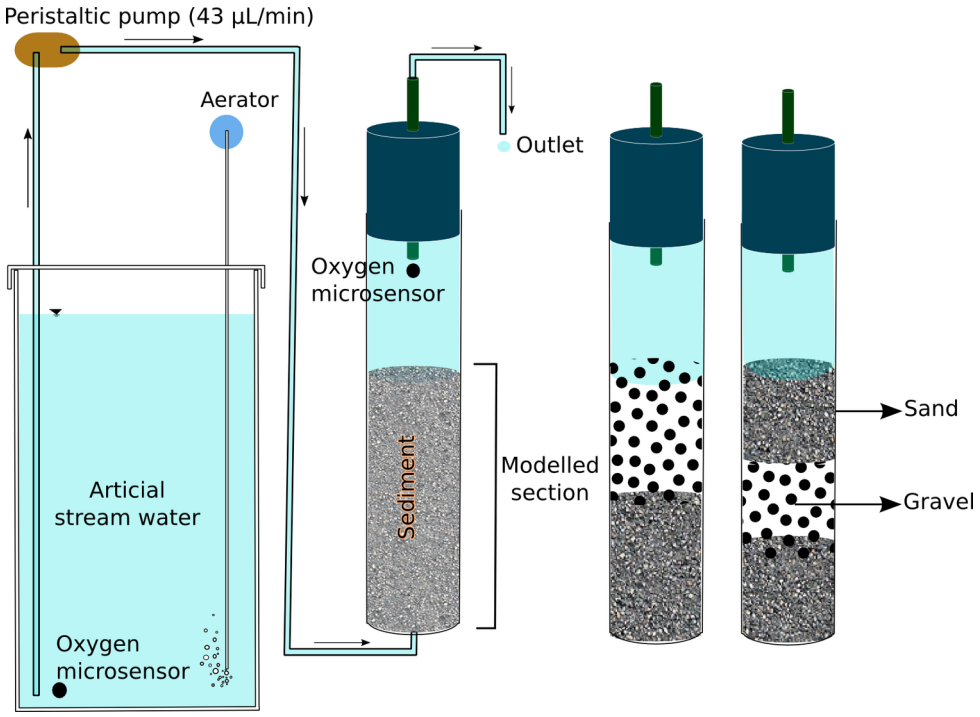

Figure 5.1: Schematic representation of the laboratory model set-up. (Source: Notay et al. (2014))

measurement consisting of three replicates. These arrangements are subsequently referred to as S for microcosms containing only sand; G for microcosms containing only gravel; SG and for microcosms containing sand and gravel in equal proportions but with the former containing sand in the bottom half and the latter containing gravel in the bottom half; and finally SGS for the set-up containing sand in the bottom third, gravel in the middle third and again sand in the upper third of the microcosm (see figure 5.1). All microcosms remained saturated throughout the experiment. The experiment was allowed to run for $5\,\mathrm{d}$ and regular measurements were taken for the pumping-rate of the water and for the oxygen concentrations at the inlet and outlet of the microcosms. These were then used to calculate the respiration rate as the proxy Hz heterotrophic metabolism (Notay et al., 2014).

5.1.2 Hyporheic zone metabolism: numerical model

For the purpose of the current work, a model for simulating the aerobic metabolism in the Hz (Boulton et al., 1998) has been developed. As mentioned by Gordon et al.

(2004), metabolism is best measured by monitoring oxygen concentration. As a consequence, the model currently simulates metabolism by modelling it as a function of oxygen consumption by the sediment community in the Hz.

Governing equations

The supply of oxygen for metabolism to the microorganisms in the Hz is modelled as the transport of oxygen as a solute of water flowing through the sediments. The transport equation in this case can be written as (Hinkelmann, 2005):

$$\frac{\partial (\rho c)}{\partial t} + \nabla \cdot (\mathbf{v}\rho c - \mathbf{D}\nabla (\rho c)) = q_o \tag{5.1}$$

Here ρ is the density of water, c is the concentration of the dissolved oxygen, $\mathbf{v}$ is the Darcy velocity of water, D is the dispersion tensor for the dissolved oxygen and t is the time. The source and sink for the dissolved oxygen due to respiration q_o can be assumed to be independent from the concentration of the dissolved oxygen, since these concentrations are well above the substrate saturation constant for aerobic respiration (Sheibley et al., 2003).

Dispersion, including molecular diffusion, which is quite low ($2.2 \times 10^{-8}\,\mathrm{m^2\,s^{-1}}$), was not taken into account. Furthermore, assuming that the density of water is not dependent on the amount of dissolved oxygen, equation 5.1 becomes:

$$\frac{\partial c}{\partial t} + \nabla \cdot (\mathbf{v}c) = \frac{q_o}{\rho} \tag{5.2}$$

The spatial discretisation of the model is in the form of a cell-centred finite difference scheme with a regular structured grid. For the temporal discretisation, a backward-difference explicit Euler scheme (Ferziger and Perić, 2002) with fixed time steps is used (Notay et al., 2014):

$$c_{i,j}^{n+1} = c_{i,j}^{n} - \Delta t \left[u \frac{c_{i,j}^{n} - c_{i-1,j}^{n}}{\Delta x} + v \frac{c_{i,j}^{n} - c_{i,j-1}^{n}}{\Delta y} \right] + \frac{\Delta t q_o}{\rho} \tag{5.3}$$

Where i and j are the spatial indices of the finite cell, n is the time step index, $c_{i,j}^{n}$ is the oxygen concentration in cell i, j at time step n, u and v are the flow velocities in the x and y directions respectively, q_o is the source and sink term for oxygen, Δx and Δy are the length of the finite cell in the x and y direction respectively, and Δt is the time step size.

5.1.3 Subsurface flow model: DuMu$^{\mathrm{X}}$

The current work uses DuMu$^{\mathrm{X}}$ (Flemisch et al., 2011) for simulating subsurface flow. A single-phase model (1p model) implemented by DuMu$^{\mathrm{X}}$ has been used to simulate subsurface flow under saturated conditions. For partially saturated soil, the implementation of Richards model in DuMu$^{\mathrm{X}}$ simulating simplified two-phase subsurface flow model with water and air as the wetting and non-wetting phases respectively has been used. The following section describes the mathematical models in more details.

Governing equations: Single-phase model

The continuity equation for the single-phase flow under saturated conditions can be written as (Hinkelmann, 2005):

$$\frac{\partial (\rho\phi)}{\partial t} + \nabla (\rho \mathbf{v}) = q \tag{5.4}$$

Here ρ is the density of the fluid, ϕ is the soil porosity, v is the Darcy velocity of the fluid, q is the source and sink term for the fluid and t is time.

The Darcy approach is used for the conservation of momentum (Hinkelmann, 2005):

$$\mathbf{v} = -\frac{\mathbf{K}}{\mu} (\nabla p - \rho \mathbf{g}) \tag{5.5}$$

Here $\mathbf{K}$ is the permeability of the soil, p is the pressure, μ is the dynamic viscosity of the fluid and g is the gravitational acceleration.

Combining equations 5.4 and 5.5 we arrive at:

$$\frac{\partial (\rho\phi)}{\partial t} + \nabla \left(-\rho \frac{\mathbf{K}}{\mu} (\nabla p - \rho \mathbf{g}) \right) = q \tag{5.6}$$

For the simulations performed for this work using the single-phase model, the density of water ρ and the porosity of the soil ϕ were both constant.

Equation 5.6 is solved by DuMu$^{\mathrm{X}}$ using a vertex-centred finite volume scheme as the space discretisation and a fully-implicit Euler scheme for time discretisation (DuMu$^{\mathrm{X}}$, 2014).

Governing equations: Richards model

In partially saturated soil, the saturation of phase α is defined as the ratio of the volume of the fluid α to the total pore volume, where α can either be the wetting phase w or the non-wetting phase n (Hinkelmann, 2005).

The fluid saturation for phase α is defined as the ratio of the volume of the fluid to the total pore volume of the soil (Hinkelmann, 2005).

The capillary pressure in the soil is described by Van Genuchten in terms of effective water saturation parameters α and n as follows (Hinkelmann, 2005):

$$p_c\left(S_w\right) = \frac{1}{\alpha}\left(S_e^{-\frac{1}{m}} - 1\right)^{\frac{1}{n}} \tag{5.7}$$

Here p_c is the capillary pressure, S_w is the water saturation and $m = 1 - \frac{1}{n}$. S_e is the effective water saturation and is defined as:

$$S_e\left(p_c\right) = \frac{S_w - S_{wr}}{1 - S_{wr}} \tag{5.8}$$

Where S_{wr} the residual water saturation.

The relative permeability of the soil for phase α varies with the soil saturation and is defined by the Van Genuchten as follows (Hinkelmann, 2005):

$$k_{rw} = S_e^{\frac{1}{2}}\left[1 - \left(1 - S_e^{\frac{1}{m}}\right)^m\right]^2 \tag{5.9}$$

$$k_{rn} = \left(1 - S_e\right)^{\frac{1}{3}}\left[1 - S_e^{\frac{1}{m}}\right]^{2m} \tag{5.10}$$

Where k_{rw} and k_{rn} are the relative permeabilities for the water and air phases respectively and m is described in terms of the Van Genuchten parameter n as $m = 1 - \frac{1}{n}$.

The mass conservation equation for Richards flow for the aqueous phase can be written as shown below (Hinkelmann, 2005):

$$\frac{\partial\left(\phi\rho S_w\right)}{\partial t} - \nabla\left(\rho\mathbf{v}\right) = q_w \tag{5.11}$$

Here ρ is the density of water, ϕ is the soil porosity, v is the Darcy velocity of water, q_w is the source and sink term for water and t is the time.

The generalised Darcy's law for the aqueous phase, when Reynold's number is lower than 1, can be written as (Hinkelmann, 2005):

$$\mathbf{v} = -\frac{k_{rw}}{\mu}\mathbf{K}\left(\nabla p_w - \rho\mathbf{g}\right) \tag{5.12}$$

Where μ is the dynamic viscosity of water p_w is the pressure of the water phase, $\mathbf{K}$ is the permeability of the soil and g is the gravitational acceleration.

Combining equations 5.11 and 5.12 we arrive at (Hinkelmann, 2005):

$$\frac{\partial\left(\phi\rho S_w\rho\right)}{\partial t} - \nabla\left(\rho\frac{k_{rw}}{\mu}\mathbf{K}\left(\nabla p_w - \rho\mathbf{g}\right)\right) = q_w \tag{5.13}$$

For Richards flow, assuming that the non-wetting phase is air and the pressure of this phase is constant throughout the domain (atmospheric pressure), the following two additional conditions hold (DuMu$^{\mathrm{X}}$, 2014, Hinkelmann, 2005):

$$S_w + S_n = 1 \tag{5.14}$$

$$p_{atm} - p_w = p_c \tag{5.15}$$

Equation 5.13 further leads to the Richards equation (Richards, 1931):

$$\frac{\partial\left(\phi\rho S_w\right)}{\partial t} + \nabla\left(\rho\frac{k_{rw}}{\mu}\mathbf{K}\left(\nabla p_c + \rho\mathbf{g}\right)\right) = \rho q_w \tag{5.16}$$

Where ϕ is the porosity of the soil and the remaining terms are as explained previously.

The spatial discretisation used by the model is a vertex-centred finite-volume method (box-method), while a fully-implicit Euler scheme is used for temporal discretisation (DuMu$^{\mathrm{X}}$, 2014, Stadler et al., 2012).

5.1.4 Surface flow model: HMS

One of the models used for simulating surface flow is HMS, which includes a shallow water equation solver capable of simulating problems such as overland flow, wetting and drying interfaces, super-, sub- and transcritical flow, shock-type flow discontinuities, etc. with second-order accuracy in space (Simons et al., 2014).

Governing equations

The general form of the conservation law in two dimensions can be expressed in the differential form as (Alcrudo and Garcia-Navarro, 1993):

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot \mathbf{F} = \mathbf{S} \tag{5.17}$$

Where $\mathbf{U}$ is the vector of the conserved quantities, $\mathbf{F}$ is the flux tensor and $\mathbf{S}$ is the vector of source and sink terms.

Rewriting the flux tensor $\mathbf{F}$ in the form $\mathbf{F} = (\mathbf{E}, \mathbf{G})$, equation 5.17 becomes:

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{E}}{\partial x} + \frac{\partial \mathbf{G}}{\partial y} = \mathbf{S} \tag{5.18}$$

$\mathbf{U}$, the vector of the conserved quantities can be expressed as:

$$\mathbf{U} = \begin{bmatrix} h \\ uh \\ vh \end{bmatrix} \tag{5.19}$$

Where h is the water depth, and u and v are the components of the water velocity vector in x and y directions respectively.

The cartesian components of the flux vector are expressed as:

$$\mathbf{E} = \begin{bmatrix} uh \\ u^2h + \frac{1}{2}gh^2 - \nu_t \frac{\partial (uh)}{\partial x} \\ uvh \end{bmatrix}, \mathbf{G} = \begin{bmatrix} vh \\ uvh \\ v^2h + \frac{1}{2}gh^2 - \nu_t \frac{\partial (vh)}{\partial y} \end{bmatrix} \tag{5.20}$$

Where g is the gravitational acceleration and ν_t is the turbulent kinematic viscosity.

Finally, the vector of the source terms $\mathbf{S}$ can be expressed in the form (Simons et al., 2014):

$$\mathbf{S} = \begin{bmatrix} q \\ -\frac{\tau_{Bx}}{\rho} - gh\frac{\partial z_B}{\partial x} \\ -\frac{\tau_{By}}{\rho} - gh\frac{\partial z_B}{\partial y} \end{bmatrix} \tag{5.21}$$

Here q is the source and sink term for water and z_B is the bottom elevation above the datum. The bed friction term can be written as:

$$-\frac{\tau_{\mathbf{B}}}{\rho} = -\frac{g}{C^2}|\mathbf{v}|\mathbf{v} \tag{5.22}$$

where C is either a constant value like in the original Chézy formula or be a function e.g. from Manning's formula: $C = h^{1/6}n^{-1}$ (Simons et al., 2014). For the current work, the Darcy-Weisbach friction term is used for the coupled model simulating the laboratory experiment by Smith and Woolhiser (1971) while it is ignored for other coupled models using HMS.

In space, a cell-centred finite-volume scheme is used for discretisation of the equation system described in equation 5.17 (Simons et al., 2014). Along the time axis the explicit forward Euler method is used as shown in equation 5.23 with the timestep size being adapted during the simulation based on the CFL (Courant-Friedrichs-Lewy) criterion (Simons et al., 2014).

$$\mathbf{U}^{i+1} = \mathbf{U}^i - \frac{\Delta t}{A}\sum_k \left(\mathbf{f}_k^i \cdot \hat{\mathbf{n}}_k\right)\Delta l_k + \Delta t \mathbf{S}^i \tag{5.23}$$

Here i is the time step index, k is the index of the edge within the finite volume cell, $\mathbf{U}$ is the vector of the conserved quantities, $\mathbf{S}$ is the vector of the source terms, A is the area of the finite volume cell, $\mathbf{f}$ is the flux vector through the common edge, $\hat{\mathbf{n}}$ is the outward unit vector normal to the cell edge, Δl is the length of the cell edge and Δt is the time step size.

5.1.5 Surface flow model: TELEMAC-2D

TELEMAC-MASCARET is an integrated suite of solvers for use in the field of free-surface flow. Having been used in the context of many studies throughout the world, it has become one of the major standards in its field (open TELEMAC-MASCARET, 2015). TELEMAC-2D is a module of TELEMAC-MASCARET for solving 2D shallow water equations, which is the other model that has been used for the hydrodynamic simulation of surface flow, which was already discussed in section 5.1.4.

TELEMAC-2D uses an SUPG finite element scheme for the spatial discretisation of water depths by default. Furthermore, in the default settings, an operator splitting method combining the method of characteristics with the finite element method for velocities is used. Other discretisation schemes available are the con-

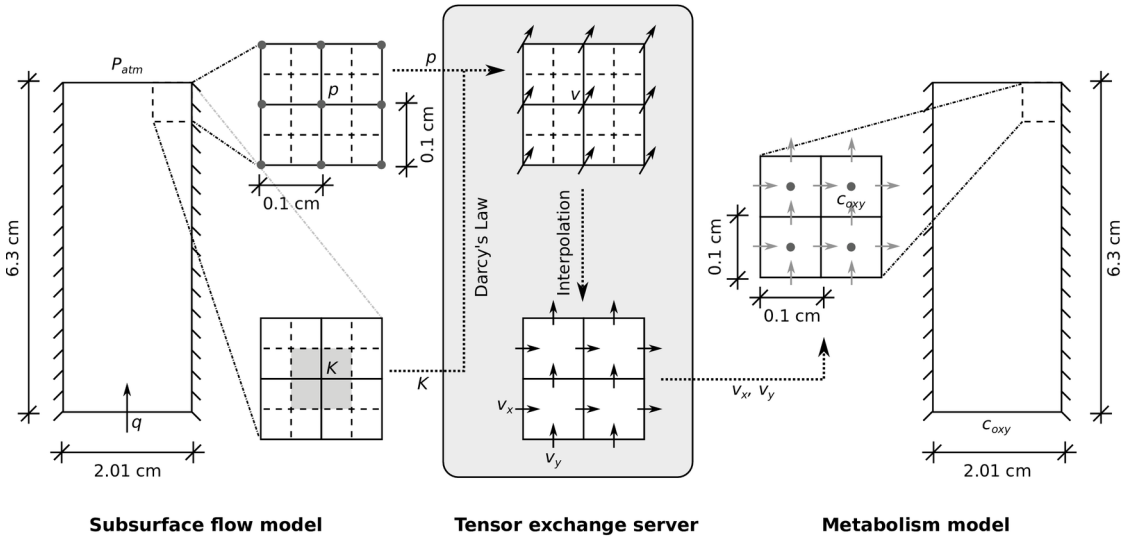

Figure 5.2: Coupled subsurface flow and metabolism model set-up

servative N-scheme, conservative PSI-scheme, non-conservative PSI-scheme, implicit non-conservative N-scheme and edge-based N-scheme. (TELEMAC-MASCARET, 2014). For the current work, only the default settings for space and time discretisation were used. Further details about the theoretical background of the TELEMAC-2D model can be found in Hervouet (2007).

5.2 Coupled subsurface flow and metabolism model

The Hz and the streambed host a great part of the stream metabolism, e.g. 97% of whole stream denitrification (Böhlke et al., 2009) and 40–93% of whole stream respiration (Fellows et al., 2001). Metabolism is one of the most integrative ecosystem-level functions in rivers since it gives clues about the energy and material fluxes through ecosystems (Enquist et al., 2003). Despite the recognition of this fact, there is still a lack of understanding of complex responses of biological processes to the hydromorophology of the Hz (Elosegi et al., 2010) and as a consequence, research tends to consider Hz as a hydrogeomorpholgically homogeneous black box. Previous research has tried to combine heterogeneity with metabolism at the reach scale, for instance distinguishing several zones withing the transient storage zone depending on their metabolic activity (Argerich et al., 2011) or differentiating different zones of the streambed based on their residence time (Choi et al., 2000). A more complex

model taking heterogeneities in the hydrogeomrophology would help clarify whether Hz heterogeneity needs to be considered when studying stream metabolism. A possible means for achieving this goal is to couple a model simulating the flow of water along with dissolved nutrients such as oxygen in the Hz. As a proof of concept of this approach, a Hz metabolism model (see section 5.1.2) simulating the transport of dissolved oxygen has been developed and coupled with a subsurface flow model. The coupled model and its initial results are presented in the following sections.

Figure 5.2 shows a schematic diagram of the set-up used for the coupled subsurface flow Hz metabolism model. For both models, the dimensions of the domain are the same as those of the microcosm in the laboratory experiment (see section 5.1.1) i.e. $2.01\,\mathrm{cm} \times 6.3\,\mathrm{cm}$. The cells of the computational grid for the subsurface flow model are also square shaped with an edge length of approximately $0.1\,\mathrm{cm}$. For the Hz metabolism model square shaped cells with an edge length of $0.1\,\mathrm{cm}$.

As initial conditions, the pressure is set to atmospheric pressure throughout the domain of the subsurface flow model, while for the Hz metabolism model, the dissolved oxygen concentration at the inflow measured in the laboratory ($8.55\,\mathrm{mg\,l^{-1}}$) was used as the initial condition throughout the domain. For the subsurface flow model, a Neumann boundary condition using the flow rate measured in the laboratory ($2.58\,\mathrm{ml\,h^{-1}}$) was set as the lower boundary condition while at the upper boundary, the atmospheric pressure was set as the Dirichlet boundary condition. For the Hz metabolism model, the dissolved oxygen concentration measured in the laboratory ($8.55\,\mathrm{mg\,l^{-1}}$) was set as the lower boundary condition while the upper boundary was set as an open boundary condition ($\frac{\partial c}{\partial z} = 0$). The lateral boundaries were treated as closed for both the models. The sink term for the dissolved oxygen was set to $9.88 \times 10^{-11}\,\mathrm{g\,m^{-3}\,s^{-1}}$ for gravel and $1.48 \times 10^{-11}\,\mathrm{g\,m^{-3}\,s^{-1}}$ for sand from the laboratory measurements. Since the boundary conditions and the source and sink terms are constant in time, the coupled simulation reaches a steady state soon after starting.

It should be noted that even when the cells of the subsurface flow and Hz metabolism model overlap completely, spatial interpolation is still required when transferring information from one model to the other. This is because the pressure fields calculated by the subsurface flow model are at the vertices, while for the calculation of fluxes, the Hz metabolism model requires velocities at the edge-centres (see figure 5.2). The TES (see section 4.3) handles this interpolation, as well as the interpolation in time and the calculation of the velocity field from the pressure field.

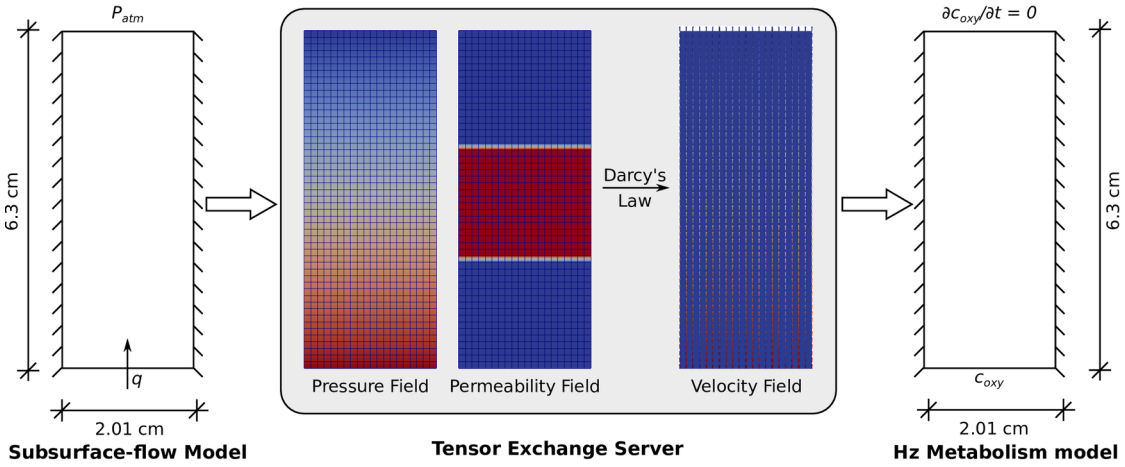

Figure 5.3: Schematic diagram of coupled subsurface flow and Hz metabolism model

5.2.1 Coupling mechanism

Figure 5.3 shows a schematic diagram of the coupling mechanism for the coupled subsurface flow and Hz metabolism model. Here:

- The subsurface flow model is responsible for the computing the pressure field of the fluid in the subsurface (see section 5.1.3).
- The Hz metabolism model simulates the metabolism processes in the subsurface by simulating the transport of dissolved oxygen in the water (see section 5.1.2) but in order to do so, it requires the velocity field from the subsurface model.
- The Tensor Exchange Server (TES) plays a very important role in the coupling process and is responsible for:
  - coupling models written in different programming languages, namely C++ for DuMu^{X} and Java for the Hz metabolism model,
  - handling the communication between the models,
  - applying Darcy's law for computing the velocity field from the pressure field and
  - for interpolating the velocities in both space and time.

The communication between the TES and the individual models takes place through XML-RPC. It can also be noted from figure 5.3 that the flow of information is uni-

directional i.e. from the subsurface flow model to the TES to the Hz metabolism model. Another point to note here is that the coupled interface is two-dimensional.

Algorithms 1 and 2 explain the roles of the subsurface flow and Hz metabolism models in the coupling process respectively. The TES has two processes running in parallel for serving the two models. The *modus operandi* of these two processes is shown in algorithms 3 and 4.

Algorithm 1 Algorithm for subsurface flow model

$n \leftarrow 1$
$t_d \leftarrow 0$
Send $\mathbf{K}(x_d)$ to TES
while $t_d < t_{end}$ **do**
    Calculate n-th time step
    Send t_d and $p(x_d, t_d)$ to TES
    $n \leftarrow n + 1$
    $t_d \leftarrow t_d + \Delta t_d$
end while

Algorithm 2 Algorithm for Hz metabolism model

$n \leftarrow 1$
$t_m \leftarrow 0$
while $t_m < t_{end}$ **do**
    Request $\mathbf{v}(x_m, t_m)$ from TES
    Calculate n-th time step
    $n \leftarrow n + 1$
    $t_m \leftarrow t_m + \Delta t_m$
end while

Algorithm 3 Interactions between TES and subsurface flow model

$t_{dlast} \leftarrow 0$
Receive $\mathbf{K}(x_d)$ from subsurface flow model
while Simulation is running **do**
    Receive t_d and $p(x_d, t_d)$ from subsurface flow model
    $t_{dlast} \leftarrow t_d$
    Calculate $\mathbf{v}(x_d, t_d)$ using Darcy's law
end while

Figure 5.4 summarises the progress of the coupled simulation and the interactions between the TES and the coupled models with the help of a UML diagram.

Algorithm 4 Interactions between TES and the Hz metabolism model

1: $t_{dlast} \leftarrow 0$
2: **while** Simulation is running **do**
3: Receive request for $\mathbf{v}\,(x_m, t_m)$ from the metabolism model
4: **while** $t_m < t_{dlast}$ **do** ▷ see line 5 of algorithm 3
5: Wait for $p\,(x_d, t_d)$ for next value of t_d
6: **end while**
7: Interpolate $\mathbf{v}\,(x_m, t_m)$ from $\mathbf{v}\,(x_d, t_d)$ ▷ see line 6 of algorithm 3
8: **return** $\mathbf{v}\,(x_m, t_m)$ to metabolism model
9: **end while**

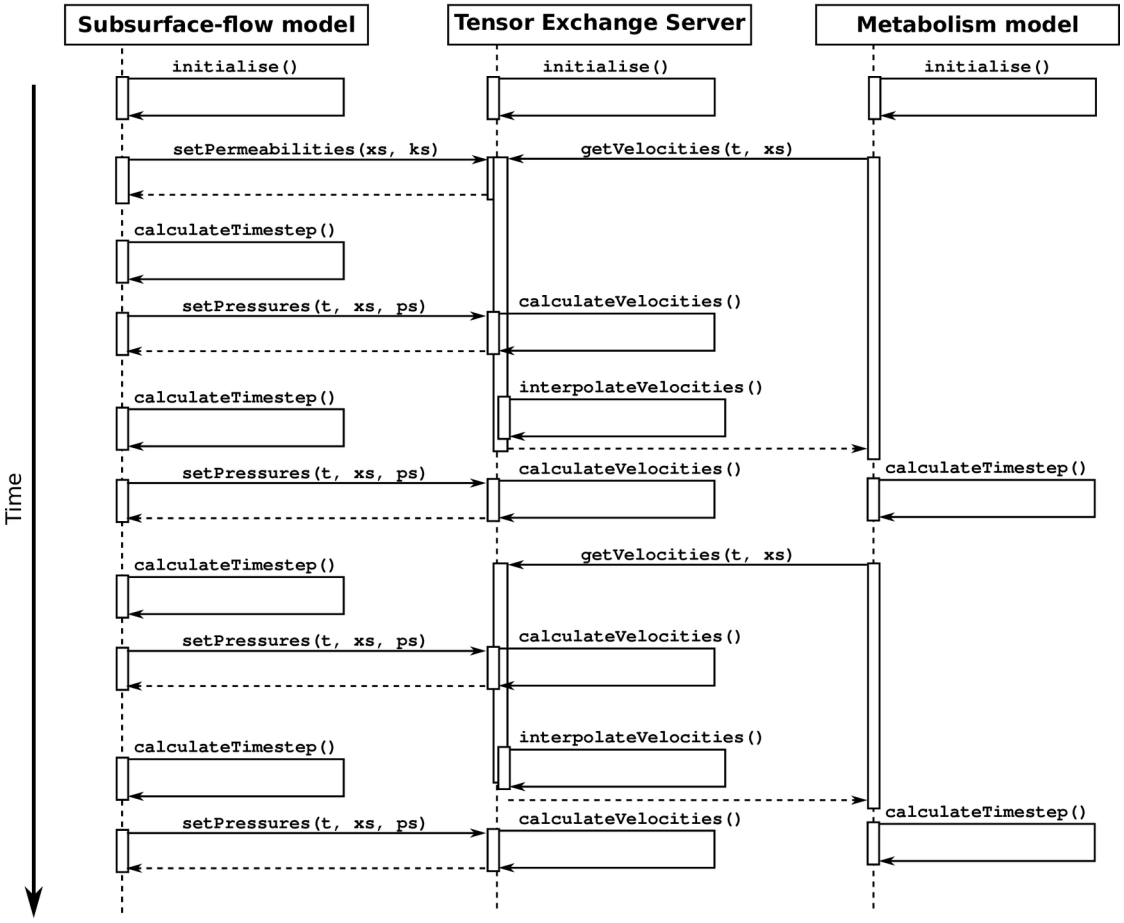

Figure 5.4: UML representation of coupled interactions for the coupled subsurface flow and metabolism model

5.2.2 Modifications required to couple models

Listing 1 in the appendix shows the necessary modifications in the C++ code that need to be made to the single-phase subsurface flow model in DuMu^{X} in order to be able to use it in a coupled simulation. As can be seen from this listing, it is necessary to modify only one class in DuMu^{X} in order to make it ready for coupling. Additionally, it is necessary to link the programme with the XML-RPC shared library during compilation. Under Debian Linux with the `libxmlrpc-c3-dev` package installed, it is necessary to add lines 4 to 6 and to include the relevant XML-RPC header files in order to use DuMu^{X} as an XML-RPC client and to communicate with the coupling broker (discussed in the next section). In this particular case, since the spatial discretisation and permeabilities don't change during the simulation, they are communicated to the server only once during the simulation. This is done in the constructor of the class being modified (lines 41 to 136). The communication with the coupling broker in order to communicate the geometry/topology and the permeability information takes place in lines 118 and 134 respectively.

The `postTimeStep` method (lines 143 to 146 of listing 1 in the appendix) provided by the single-phase model in DuMu^{X} is used to send the pressure field to the coupling broker at the end of each time step. As can be seen in this listing, the `postTimeStep` method delegates this task to the `sendPressures_` method (lines 155 to 202), which iterates over the grid to query the pressure values at each computational node and sends the pressure field to the coupling broker (line 200).

Listing 2 in the appendix shows the part of the Java code that handles the coupling in the Hyporheic zone (Hz) metabolism model. As can be seen in this listing, the model requires the velocities from the coupling broker once during each time step. It delegates this functionality to a `CouplingClient` instance (lines 10 and 23) by calling its `getVelocities` method within the time loop.

The implementation of the `CouplingClient` class is shown in listing 3 in the appendix. Similar to the XML-RPC server shown in listing 4.1, this class uses the Apache XML-RPC library in order to be able to function as an XML-RPC client. Not surprisingly, the code for enabling XML-RPC client support is similar to the code for enabling XML-RPC server support, which involves importing the relevant classes from the Apache XML-RPC library (lines 8 to 11) and configuring the client (lines 25 to 43). Like in the case of the XML-RPC server, the Apache XML-RPC library needs to be added to the classpath of the Java application. The `getVelocities` method (lines 45 to 75) is then responsible for executing the XML-RPC request, processing

the response from the XML-RPC server (coupling broker) and returning the velocity field to the Hz metabolism model in order to be able to calculate the next time step.

5.2.3 Implementation of the coupling broker

The Tensor Exchange Server (TES) acts as the coupling broker and its implementation is based on the description in section 4.3. The XML-RPC server, which is responsible for listening for XML-RPC requests from the coupled models on a predefined port, is exactly same as in listing 4.1. The `CouplingHandler`, `TensorSet` and `CouplingTensor` implementations are adapted from the classes described in section 4.3 to suit the requirements of this particular coupling scenario. Listing 4 in the appendix shows the implementation of the `CouplingHandler` for the coupling of DuMu$^{\text{X}}$ and the Hz metabolism model. For coupling these two particular models, the public methods `setCoordinates` (lines 9 to 27), `setPermeabilities` (lines 29 to 41) and `setPressures` (lines 43 to 56) of the `CouplingHandler` class handle the communication with DuMu$^{\text{X}}$ for setting the geometry/topology, permeability field and pressure field for the given time respectively. Similarly, the method `getVelocities` (lines 58 to 78) handles the communication with the Hz metabolism model and returns the velocity field at the time requested by the model.

Listings 5 and 6 in the appendix show the implementation of the `CouplingTensor` and `TensorSet` classes in Java for the coupling of DuMu$^{\text{X}}$ with the Hz metabolism model. The `CouplingTensor` implementation is an adaptation of the `CouplingTensor` class shown in listing 4.3. Since the coupling involves more than one physical state variable and hence more than one tensor object, it is possible to define the units of the physical state variable that the tensor object represents at the time of construction (line 30 of listing 5 in the appendix). Additionally, it is possible to define the geometry and topology information at the time of creation of the tensor objects (lines 31 and 32 of listing 5 in the appendix). The reason for this is that the spatial discretisation used by the tensor object's grid is the same as that used by DuMu$^{\text{X}}$. The computed velocity fields are later mapped to the spatial discretisation provided by the Hz metabolism model (lines 99 to 112 of listing 6 in the appendix). The TES also uses the `CouplingTensor` implementation to make the coupled models wait until the values they require are available (lines 77 to 83 of listing 5 in the appendix). Since the `CouplingTensor` knows about the latest time step for which it has values, this is the best option to test the availability of the requested values. The `TensorSet` implementation contains references to all the

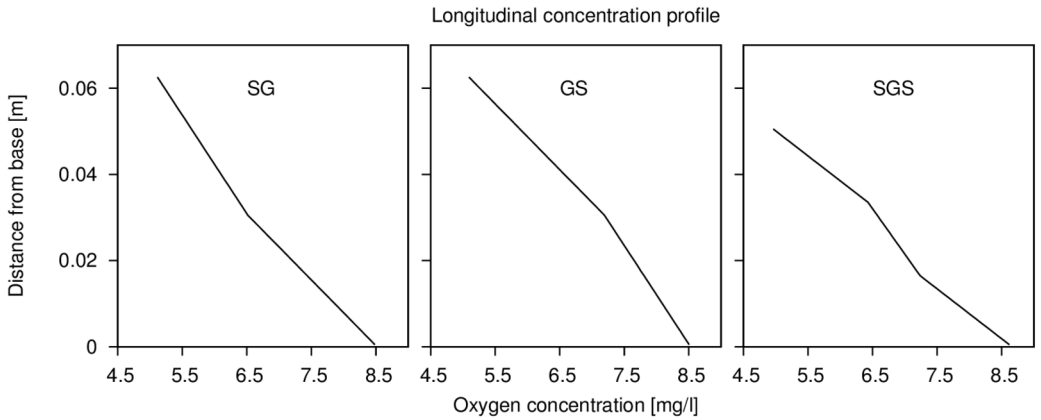

Figure 5.5: Vertical profile of the oxygen concentration distribution in the microcosm after steady state has been achieved

tensors (lines 8 to 11 of listing 6 in the appendix) relevant to the coupling of the two models and also the operators (lines 13 to 17 of listing 6 in the appendix) on which it relies for the application of Darcy's law (lines 87 to 96 of listing 6 in the appendix) and for the interpolation of velocities in space and time (lines 99 to 112 of listing 6 in the appendix).

5.2.4 Coupled simulation results

As seen in figure 5.5, the coupled model is able to capture the heterogeneities in metabolism in the microcosm arising due to the variation in permeability. Figure 5.6 shows a comparison of the oxygen concentrations measured at the outlet of the microcosm in the laboratory to the results obtained from the coupled simulation after steady-state has been achieved. As can be observed from this figure, the model is able to simulate the homogeneous set-ups (S & G) quite accurately. However, the respiration rates for the heterogeneous cases are underestimated by the coupled simulation model resulting in higher oxygen concentrations being computed at the outlet. Even for the two cases (SG and SGS) showing the largest differences from the measurement values, the simulation results lie very close to the range of values observed in the laboratory. This might be the result of unavoidable natural heterogeneities in the sediment distribution in microcosms with similar compositions. The results could perhaps be improved by repeating the experiment with more replic-

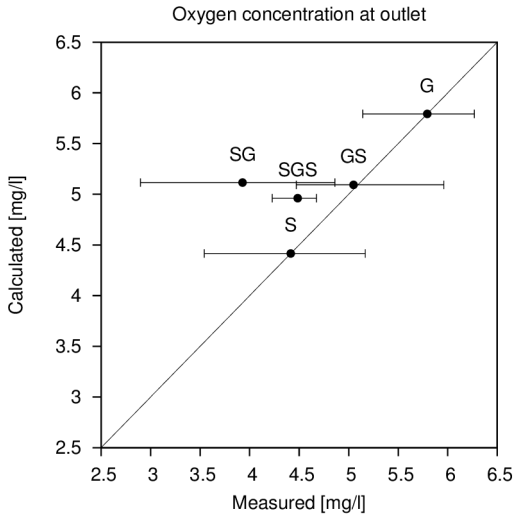

Figure 5.6: Comparison of oxygen concentrations at the outlet of microcosm measured in the laboratory (mean of measured values $\pm$ min/max values, $n = 3$) with the computed values by the coupled model after steady state is achieved. The diagonal line indicates equal values. S = Sand, G = Gravel, SG = Sand-Gravel, GS = Gravel-Sand, SGS = Sand-Gravel-Sand.

ates. Also, the current model is quite rudimentary in nature and in order to be able to better simulate the metabolic processes, it would be useful to include other processes such as diffusion and dispersion, anaerobic respiration, additional indicators of metabolism such as dissolved carbon or nitrogen compounds, etc. into account.

5.2.5 Summary

This application example presented a proof of concept for a coupled model, where tensor objects play an important role, applied to an ecological problem. This demonstrates the feasibility of applying tensor objects to problems similar to the understanding of nutrient fluxes and metabolic processes in the Hz. Using the TES, it was possible to couple models written in different programming languages (C++ and Java) and also to adapt the information according to the needs of the models through the transformation of physical state variables by the application of a physical law (Darcy's law), mapping, interpolation in space and time, etc.

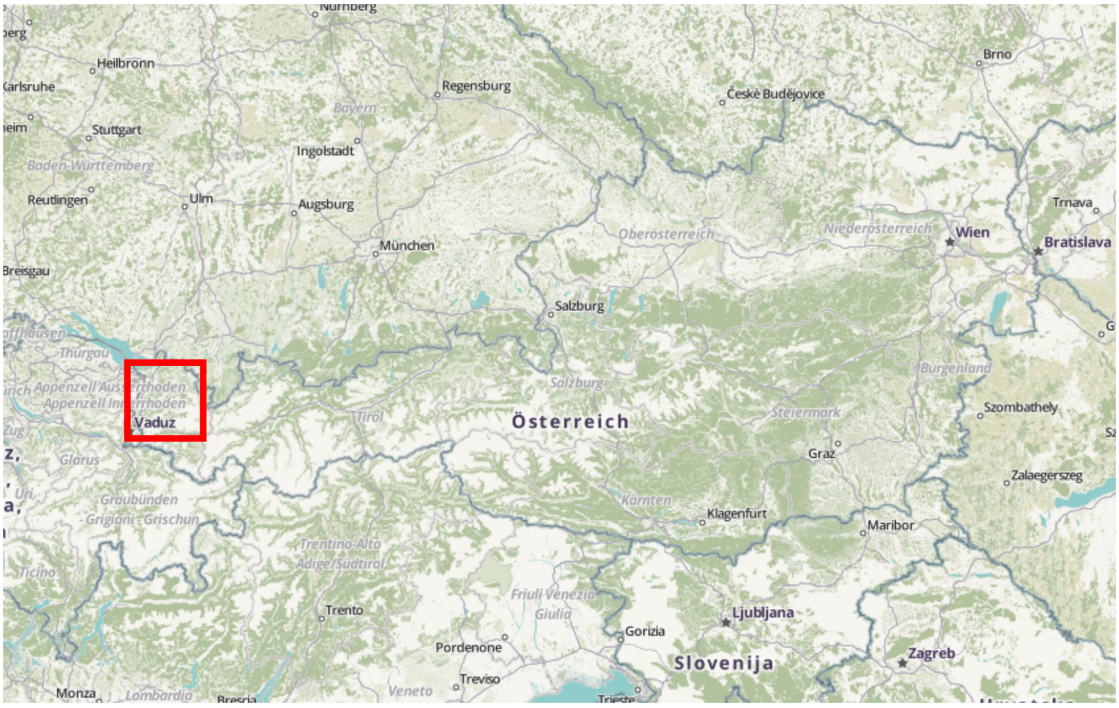

Figure 5.7: Location of the study area for the DFG Research Unit “Natural Slope”.

5.3 Coupled subsurface and surface flow model

The DFG Research Group “Natural Slope” (Hinkelmann and Zehe, 2006, 2013) was a interdisciplinary research unit investigating a slowly creeping landslide in the Heumöser slope near the town Ebnit, in the Austrian Alps (see figures 5.8 and 5.7). The project consisted of eight sub-projects working on hydrology, hydraulics, continuum mechanics, geophysics and hydroinformatics to try to better understand and explain the physical phenomena that are involved in such slow-creeping landslides. One of the proposed methods for doing this was coupling of hydroinformatic models. This included the coupling of an overland flow model with a subsurface flow model in order to investigate the effect of fast infiltration processes on the movement of the mountain slope. With this purpose in mind a coupling of the subsurface flow model DuMu^{X} with the surface flow model HMS using tensor objects was attempted. This coupling of DuMu^{X} with HMS is presented in the following sections.

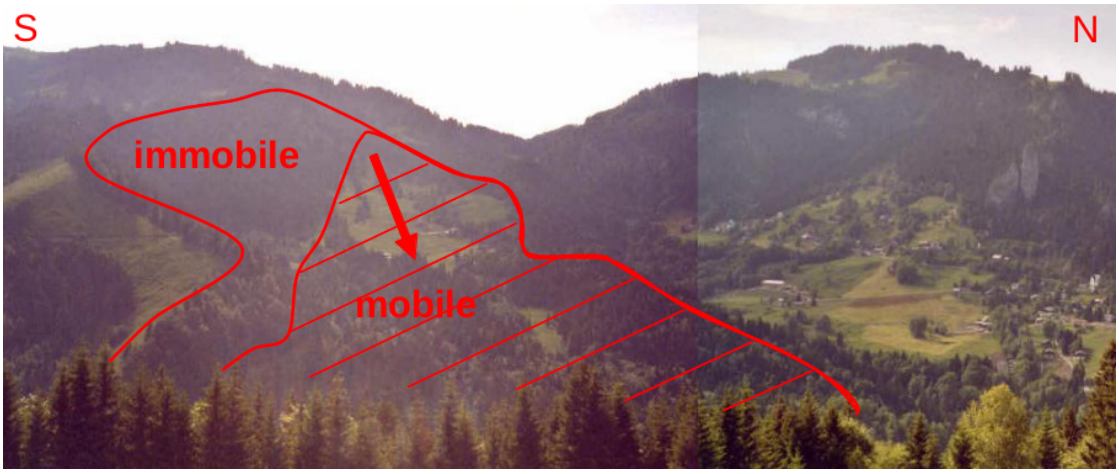

Figure 5.8: Slowly creeping landslide in the Heumöser slope. Source: Hinkelmann and Zehe (2013)

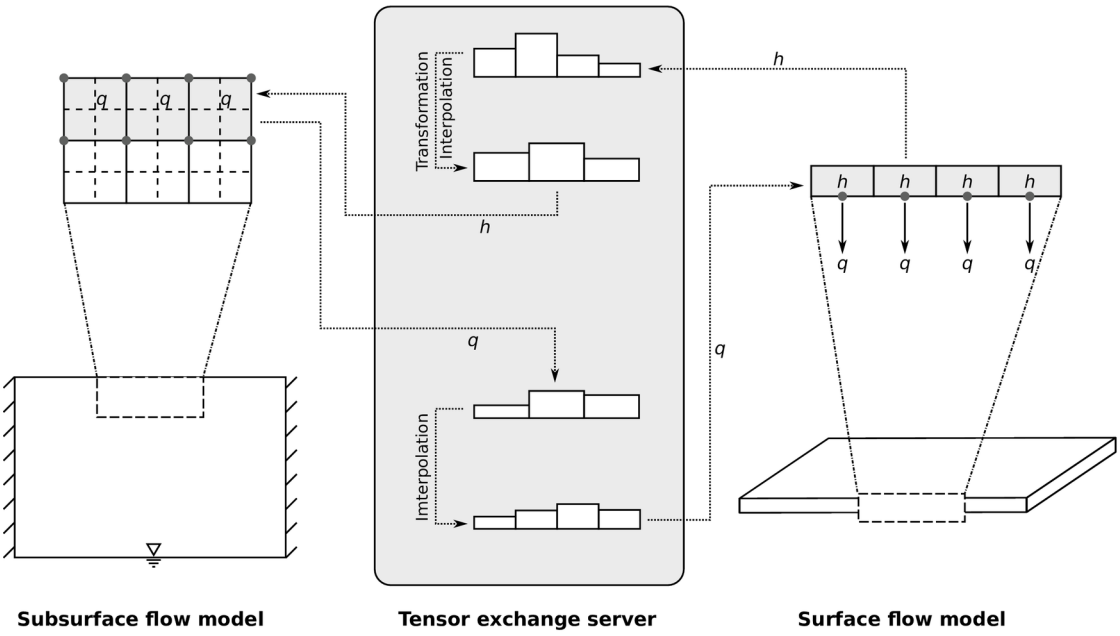

Figure 5.9: Schematic diagram of the coupled surface- and subsurface flow model.

Algorithm 5 Subsurface flow model in coupled set-up

1: $n \leftarrow 1$
2: $t_d \leftarrow 0$
3: **while** $t_d < t_{end}$ **do**
4: Request $h\left(x_d, t_d\right)$ from TES
5: Receive $h\left(x_d, t_d\right)$ from TES
6: Compute $q_{avail.}\left(x_d, t_d\right)$ from $h\left(x_d, t_d\right)$ and Δt_d
7: Compute $q_{poss.}\left(x_d, t_d\right)$
8: $q\left(x_d, t_d\right) \leftarrow min\left(q_{poss.}\left(x_d, t_d\right), q_{avail.}\left(x_d, t_d\right)\right)$
9: Calculate n-th time step
10: Send t_d and $q\left(x_d, t_d\right)$ to TES
11: $n \leftarrow n + 1$
12: $t_d \leftarrow t_d + \Delta t_d$
13: **end while**

Algorithm 6 Surface flow model in coupled set-up

1: $n \leftarrow 1$
2: $t_h \leftarrow 0$
3: **while** $t_h < t_{end}$ **do**
4: Send t_H and $h\left(x_H, t_H\right)$ to TES
5: Request $q\left(x_H, t_H\right)$ from TES
6: Receive $q\left(x_H, t_H\right)$ from TES
7: Calculate n-th time step
8: $n \leftarrow n + 1$
9: $t_H \leftarrow t_H + \Delta t_H$
10: **end while**

Algorithm 7 Interactions between TES and subsurface flow model

1: $t_{dlast} \leftarrow 0$
2: $t_{Hlast} \leftarrow 0$
3: **while** Simulation is running **do**
4: Receive t_d and $q\left(x_d, t_d\right)$ from subsurface flow model
5: $t_{dlast} \leftarrow t_d$
6: **while** $t_d < t_{Hlast}$ **do** ▷ see line 5 of algorithm 8
7: Wait for $h\left(x_H, t_H\right)$ for next value of t_H
8: **end while**
9: Interpolate $h\left(x_d, t_d\right)$ from $h\left(x_H, t_H\right)$
10: **return** $h\left(x_d, t_d\right)$
11: **end while**

Algorithm 8 Interactions between TES and surface flow model

1: $t_{dlast} \leftarrow 0$
2: $t_{Hlast} \leftarrow 0$
3: **while** Simulation is running **do**
4: Receive t_H and $h\left(x_H, t_H\right)$ from surface flow model
5: $t_{Hlast} \leftarrow t_H$
6: **while** $t_H < t_{dlast}$ **do** ▷ see line 5 of algorithm 7
7: Wait for $q\left(x_d, t_d\right)$ for next value of t_d
8: **end while**
9: Interpolate $q\left(x_H, t_H\right)$ from $q\left(x_d, t_d\right)$
10: **return** $q\left(x_H, t_H\right)$
11: **end while**

5.3.1 Coupling mechanism

Different approaches for coupling surface and subsurface flows exist. Delfs et al. (2009) have categorised these into approaches where:

1. The overland flow equation is treated as a source/sink term in the soil water balance equation.

2. The overland water depth acts as a transient boundary condition for soil water balance equation. The resulting soil water flux provides a source/sink term for the overland flow equation.

3. Flux through a thin interface layer provides source/sink terms for both the overland flow and Richards equations.

For the current work, approach 2 from the list above is used for the test cases described in section 5.3.4 while an adaptation of the third approach where the topmost row of cells acts as the interface layer is used for reproducing the laboratory experiment performed by Smith and Woolhiser (see section 5.3.5). Thus the sink term for the wetting phase in DuMu$^{\text{X}}$ in equation 5.13 is equal to the additive inverse of the exchange flux, while that for HMS in equation 5.21 is the difference of the exchange flux and the rainfall rate.

The algorithms for the coupled simulation runs by DuMu$^{\text{X}}$ and HMS are summarised in algorithms 5 and 6. Like in the case of the coupled subsurface flow and Hz metabolism model, the TES couples two models that have been developed in different programming languages, namely C++ for DuMu$^{\text{X}}$ and Java for HMS. For communicating with the models, the TES has two processes running in parallel for serving

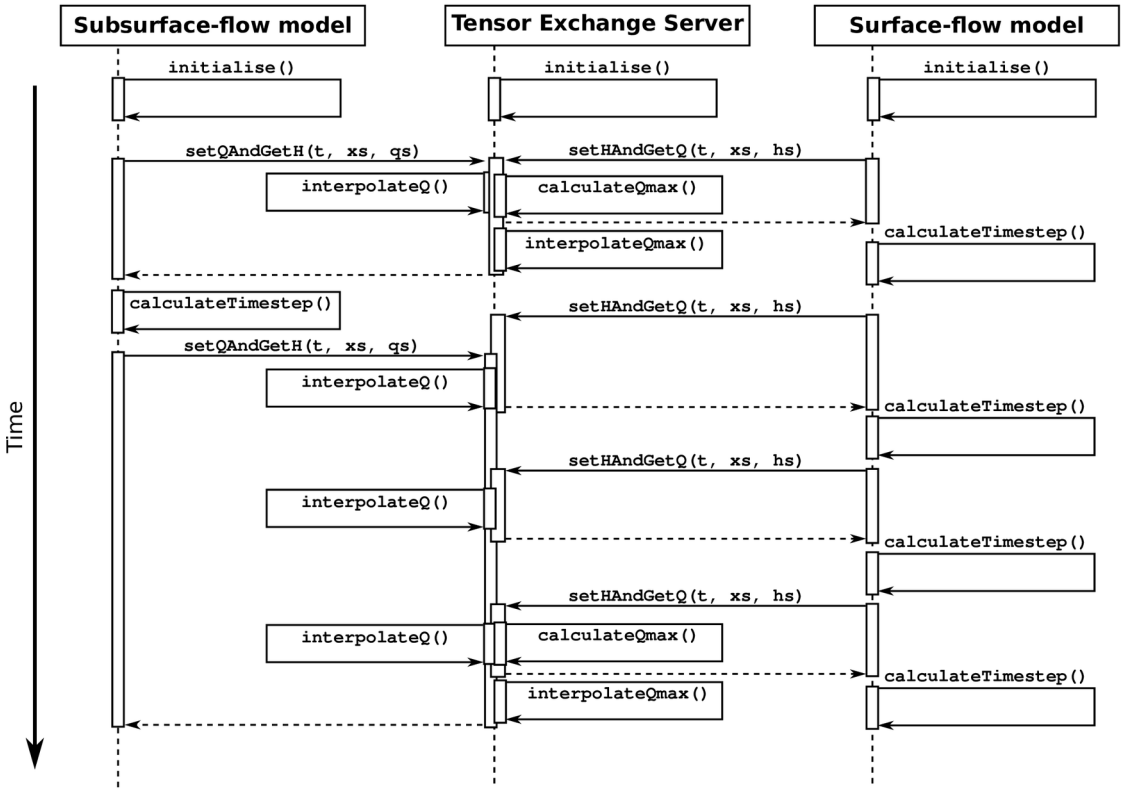

Figure 5.10: Schematic diagram of coupled subsurface flow and surface flow model

DuMu^{X} and HMS. However, an important difference in this case is that the flow of information is bidirectional, with information flowing from DuMu^{X} to HMS via TES and vice versa. The algorithms for these processes are summarised in algorithms 7 and 8 respectively. Figure 5.10 shows a UML representation of the progress of the coupled simulation with time. Figure 5.9 shows a visual representation of the role of the TES as the coupling broker and the interactions between the TES and the coupled models. The *modus operandi* of the coupled simulation is as follows:

- At the beginning of its time step, HMS sends the water depths throughout the domain to the TES.
- On receiving the request for the water depths from DuMu^{X} , the TES:
  - waits until water depths are available for a time equal to or higher than for which they are required by DuMu^{X}

  - interpolates these values in time and space to adapt them to the space and time discretisation of DuMu^{X}
  - returns the interpolated water depths to DuMu^{X}

- DuMu^{X} receives the water depths from the TES and calculates the actual exchange flux as the smaller value of the maximum available exchange fluxes, computed from the water depths received from the TES, and the possible exchange fluxes, which are computed on the basis of the conditions for the current time step.
- DuMu^{X} calculates the next time step and sends the exchange fluxes computed in the previous time step back to the TES.
- The TES saves the received values from DuMu^{X} . If the TES doesn't already has values available for a time that is equal to or greater than the time for which HMS has requested the exchange flux values, then it waits until simulation in DuMu^{X} has advanced that far and these values have been made available to the TES by it.
- The TES adapts the exchange fluxes into values that can be used by HMS by performing interpolation in space and time.
- The TES returns the computed exchange fluxes to HMS.
- HMS computes the next time step and the whole process repeats itself until the simulation ends.

5.3.2 Modifications required to couple models

The modifications necessary to be made to the C++ code to adapt the Richards model in DuMu^{X} for coupling are shown in listing 7 in the appendix. These modifications are quite similar to the ones made to the single-phase model, already discussed in section 5.2.2. Due to this reason, these modifications are touched upon only briefly here. Similar to the case for the single-phase model, the XML-RPC client functionality is added to the Richard's model by linking it against the libraries provided by the `libxmlrpc-c3-dev` package under Debian and importing the relevant header files (lines 2 to 4). At the end of each time step, the `postTimeStep` method (lines 64 to 68) computes the exchange fluxes across the computational elements along the coupling interface (`computeFluxes_` method in lines 151 to 188) and sends them

to the TES through XML-RPC (`doRPC_` method in lines 196 to 233). As a response from the TES, the model receives the interpolated waterlevels at the beginning of the next time step, which it uses for the computation of the exchange fluxes during the next time step. This computation has been programmed to be performed in the `source` method (lines **??** to 136) provided by DuMu^{X} .

In the case of HMS, a specialised surface flow layer, the `CoupledLayer` has been created using Java by overriding the `HCalcLayer` provided by HMS. The surface flow model is set up using this layer (lines 15 and 51 to 58 of listing 8 in the appendix) and it acts as the interface for querying the exchange flux to be considered as the sink/source term at each time step (lines 35 to 46 of listing 8 in the appendix).

Listing 9 in the appendix shows the Java implementation of the `CoupledLayer`. This class overrides the `compute` method (lines 88 to 107) of the `HCalcLayer`. It is thus capable of communicating with the TES for exchanging the waterdepth and exchange flux values along the coupling interface at the beginning of each time step before handing over control back to its superclass (line 106) for the actual computation of the time step. As mentioned in the previous paragraph, this class is queried for the magnitude of the exchange flux for each element at each time step. It gets these values from the TES, as is shown in lines 137 to 172 of listing 9 in the appendix. When communicating with the TES to request the fluxes at the start of a time step, the `CoupledLayer` also sends the waterdepths at the beginning of the time step along the coupling interface to the TES, as shown in lines 109 to 135 of listing 9 in the appendix. The actual communication with the XML-RPC server is delegated to the `HmsXmlRpcClient` class, whose implementation is shown in listing 10 in the appendix. This class is similar in functionality to the `CouplingClient` class explained in section 5.2.2. It uses the Apache XML-RPC library for Java to create an XML-RPC client, and is ultimately responsible for communicating with the TES through XML-RPC.

5.3.3 Implementation of the coupling broker

Similar to the coupled subsurface flow and Hz metabolism model, the Tensor Exchange Server (TES) acts as the coupling broker between DuMu^{X} and HMS and is responsible for communicating with the models and to adapt the information from one model to the requirements of the other. Similar to the TES in case of the coupled subsurface flow and Hz metabolism model, it does so by implementing three classes:

- The `CouplingServer` shown in listing 4.1 for starting the XML-RPC server

and listening for XML-RPC requests from DuMu$^{\text{X}}$ and HMS.

- The `CouplingHandler` class shown in listing 11 in the appendix, which contains the public methods that can be called by the coupled models and is responsible for processing the information provided by the models and their requests for information.
- the `CouplingTensor` class shown in listing 12 in the appendix, which is an implementation of the `Tensor` interface suitable for coupling DuMu$^{\text{X}}$ and HMS.
- The `TensorSet` class shown in listing 13 in the appendix, which is a collection of all the tensor objects required for the coupling of DuMu$^{\text{X}}$ and HMS and operators for adapting the information about the physical state variables from the tensor objects to the requirements of the coupled models.

Since these classes are similar to the corresponding classes explained in section 5.2.3, they are not described in greater detail here.

5.3.4 Test cases

The coupled DuMu$^{\text{X}}$ and HMS model was tested using six different set-ups (see figure 5.11) to test that the coupling works properly for different spatial and temporal discretisations of the two coupled models. The parameters varying across the different set-ups are summarised in table 5.1, while table 5.2 lists the parameters that were common for all the set-ups.

As initial conditions, each set-up was simulated using two types of initial conditions for DuMu$^{\text{X}}$, namely:

- **dry** initial conditions where saturation of the wetting-phase (S_w^0) at the beginning of the simulation was equal to the residual saturation S_{wr}.
- **wet** initial conditions where the saturation of the wetting-phase (S_w^0) at the beginning of the simulation was set to a value of 0.7.

In the case of DuMu$^{\text{X}}$ the bottom, left and right boundaries were set as closed boundary conditions while at the upper boundary, Dirichlet boundary condition with $S_w = 1.0$ was set. For HMS, the lateral boundaries were treated as closed boundaries.

Figures 5.12, 5.13 and 5.14 show the results obtained for the advance of the moisture front in the subsurface for the set-ups 1, 2 and 3 respectively, where the soil

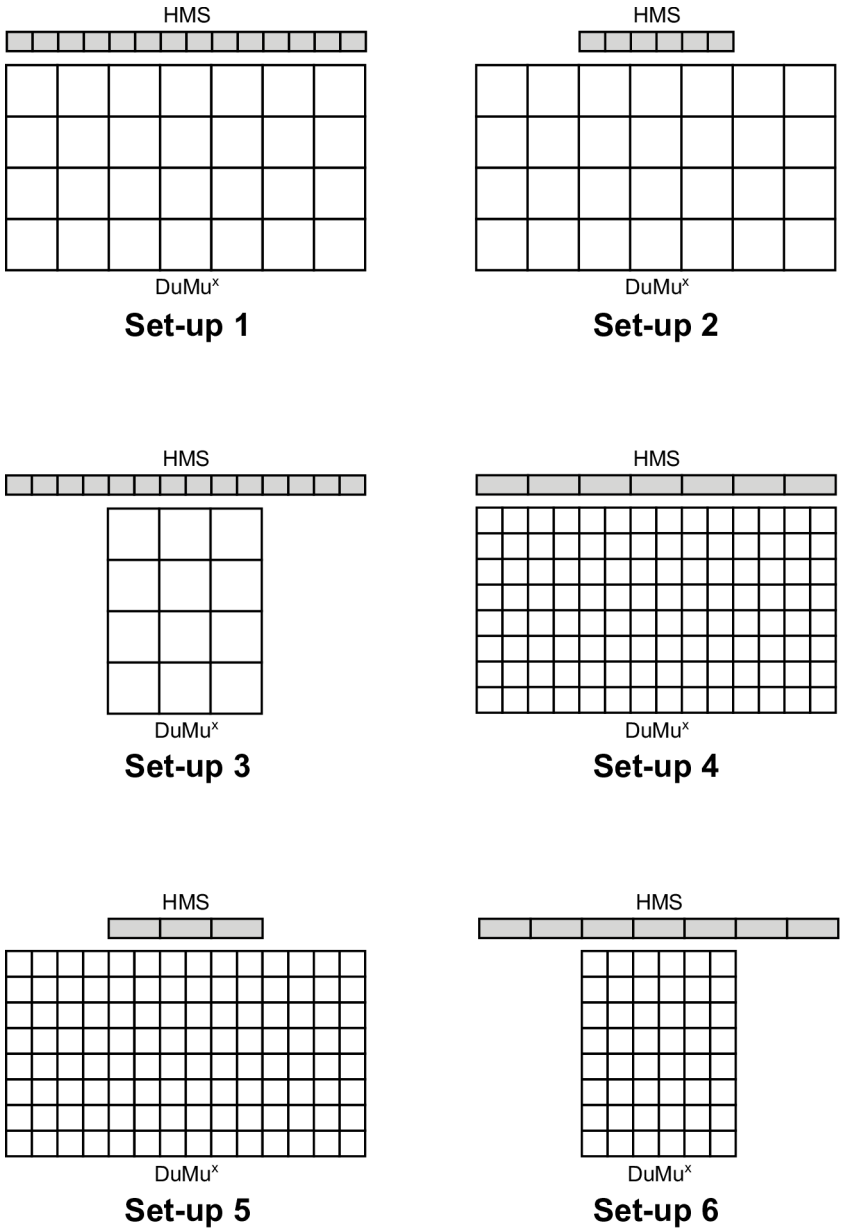

Figure 5.11: Schematic representation of the test set-ups

Parameter	**Set-up 1**	**Set-up 2**	**Set-up 3**	**Set-up 4**	**Set-up 5**	**Set-up 6**
Domain length (DuMu^{X})	5 m	5 m	2 m	5 m	5 m	2 m
Domain height (DuMu^{X})	2 m	2 m	2 m	2 m	2 m	2 m
Cell length (DuMu^{X})	0.1 m	0.1 m	0.1 m	0.05 m	0.05 m	0.05 m
Cell height (DuMu^{X})	0.1 m	0.1 m	0.1 m	0.05 m	0.05 m	0.05 m
Start of coupling interface (DuMu^{X}) at	0 m	1.5 m	0 m	0 m	1.5 m	0 m
End of coupling interface (DuMu^{X}) at	5 m	3.5 m	2 m	5 m	3.5 m	2 m
Domain length (HMS)	5 m	2 m	5 m	5 m	2 m	5 m
Domain breadth (HMS)	1 m	1 m	1 m	1 m	1 m	1 m
Cell length (HMS)	0.05 m	0.05 m	0.05 m	0.1 m	0.1 m	0.1 m
Cell breadth (HMS)	0.05 m	0.05 m	0.05 m	0.1 m	0.1 m	0.1 m
Start of coupled interface (HMS) at	0 m	0 m	1.5 m	0 m	0 m	1.5 m
End of coupled interface (HMS) at	5 m	2 m	3.5 m	5 m	2 m	3.5 m

Table 5.1: Parameters for the test set-ups

Parameter	**Value**
Soil permeability	$5.0 \times 10^{-12}\,\mathrm{m}^2$
Soil porosity	0.4
Residual water saturation	0.05
Van Genuchten α	$0.0037\,\mathrm{Pa}^{-1}$
Van Genuchten n	4.7

Table 5.2: Common parameters for all test cases

was initially dry. As can be seen from these figures, the lower edge of the saturation front advances parallel to the upper boundary of the model domain. However, for cases where the coupling interface doesn't entirely cover the upper boundary of the model domain, some lateral spreading due to capillarity can be observed in the form of rounded lateral edges of the saturation curve (see figure 5.13). Figures 5.15, 5.16 and 5.17 show the advance of the moisture front in the subsurface for the set-ups 1, 2 and 3 respectively, where the initial saturation of the water phase was 0.7. Here, the movement of the moisture, initially present in the soil, under the influence of gravity, can be noted. Similar for the cases where the initial water saturation was zero, the moisture front travels parallel to the upper boundary of the modelled domain. In cases where the coupling interface doesn't entirely cover the upper boundary of the coupled domain, the lateral spreading of the moisture front can again be observed (see figure 5.16). Additionally, the compounding of water due to the closed boundaries can be observed from these results. The conditions in the surface flow model didn't vary significantly from the initial conditions because of the way the boundary conditions are set. The water in the surface flow model is nearly stagnant, with only a very small flow introduced due by the infiltration. The results from HMS are therefore not shown here.

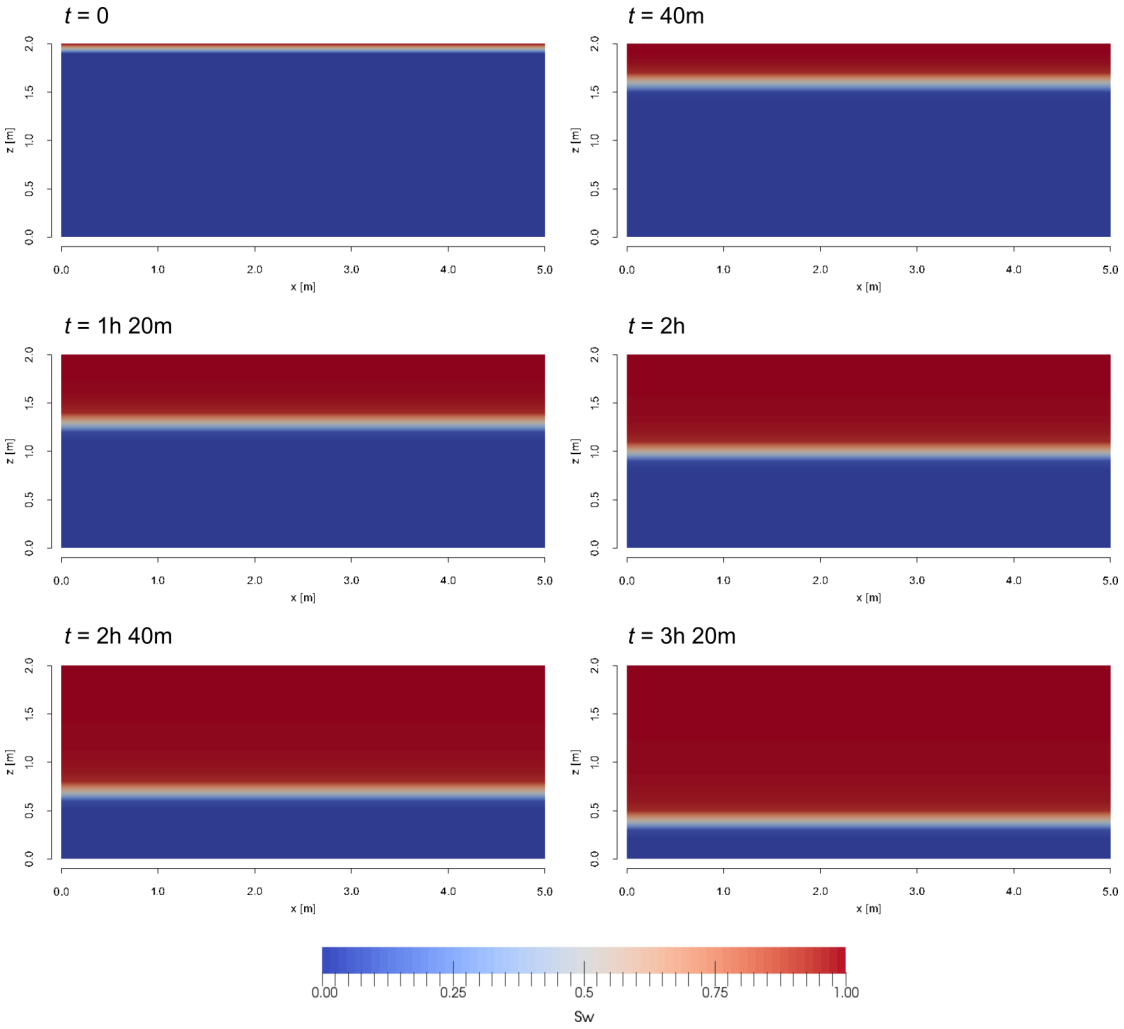

Figure 5.12: Saturation in the subsurface flow model of the coupled model for set-up 1 (dry)

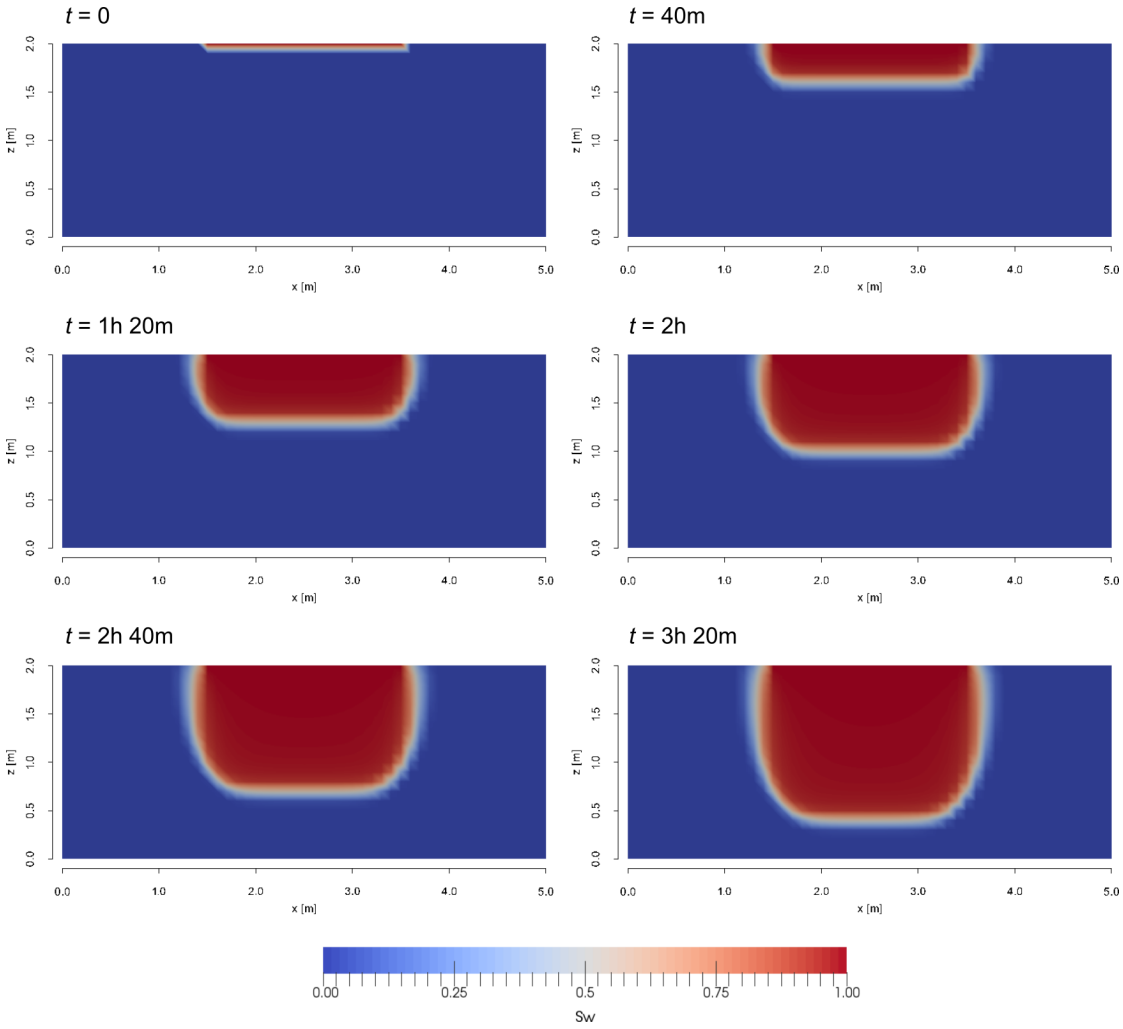

Figure 5.13: Saturation in the subsurface flow model of the coupled model for set-up 2 (dry)

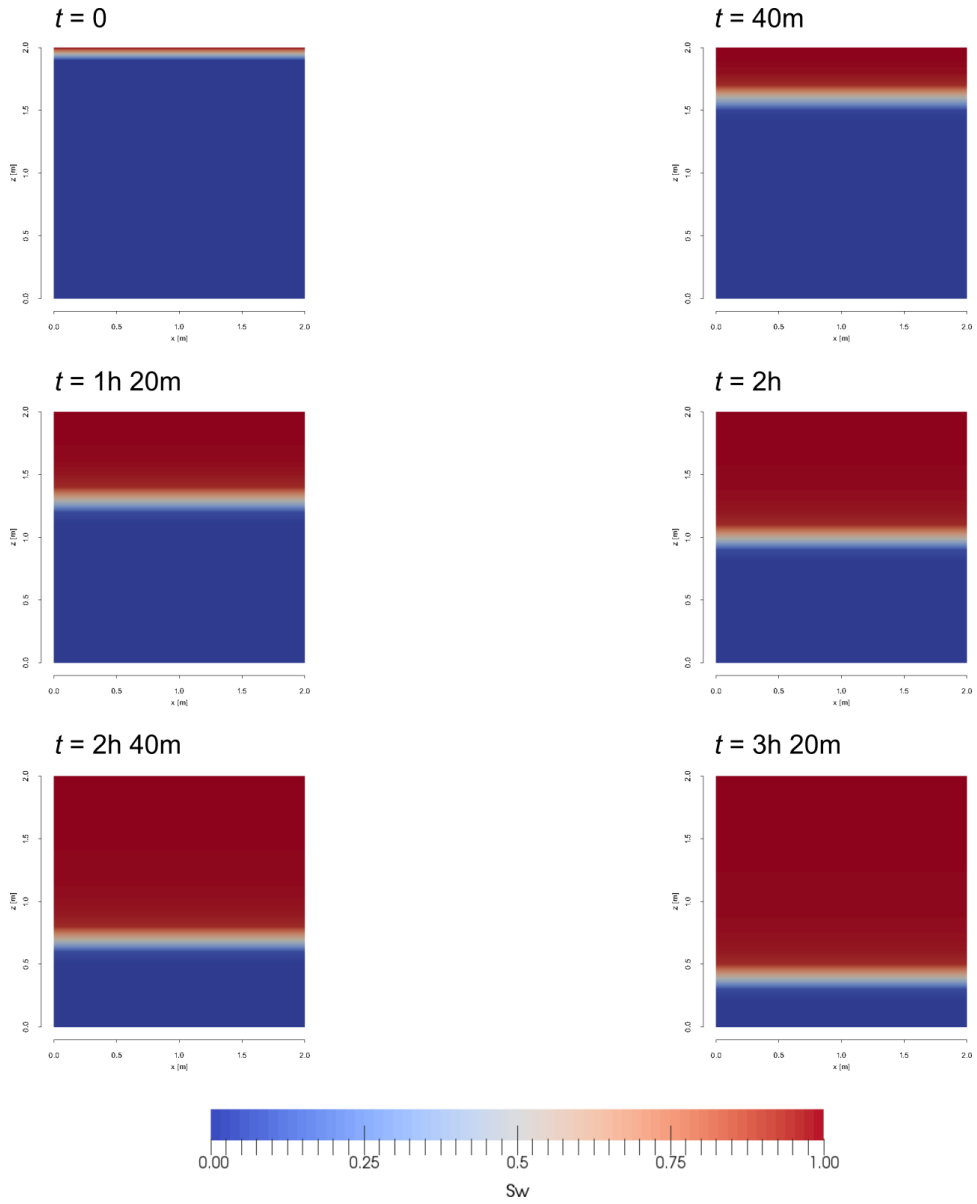

Figure 5.14: Saturation in the subsurface flow model of the coupled model for set-up 3 (dry)

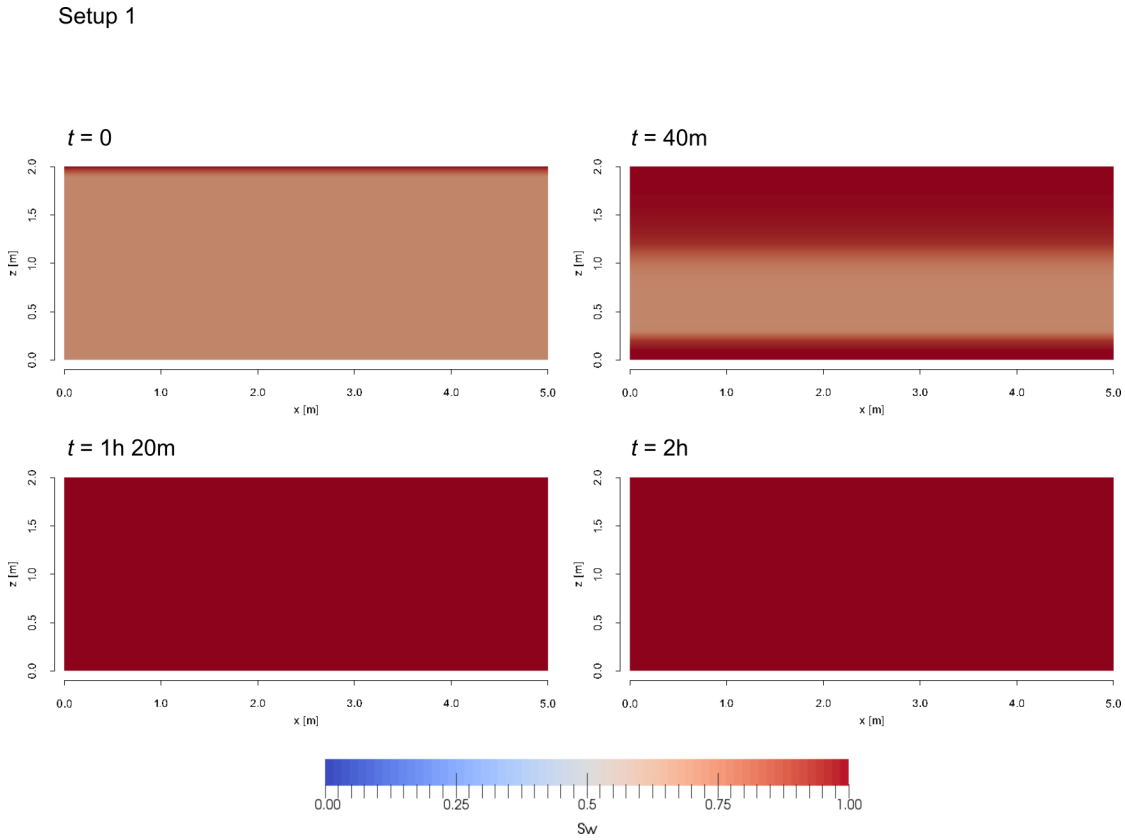

Figure 5.15: Saturation in the subsurface flow model of the coupled model for set-up 1 (wet)

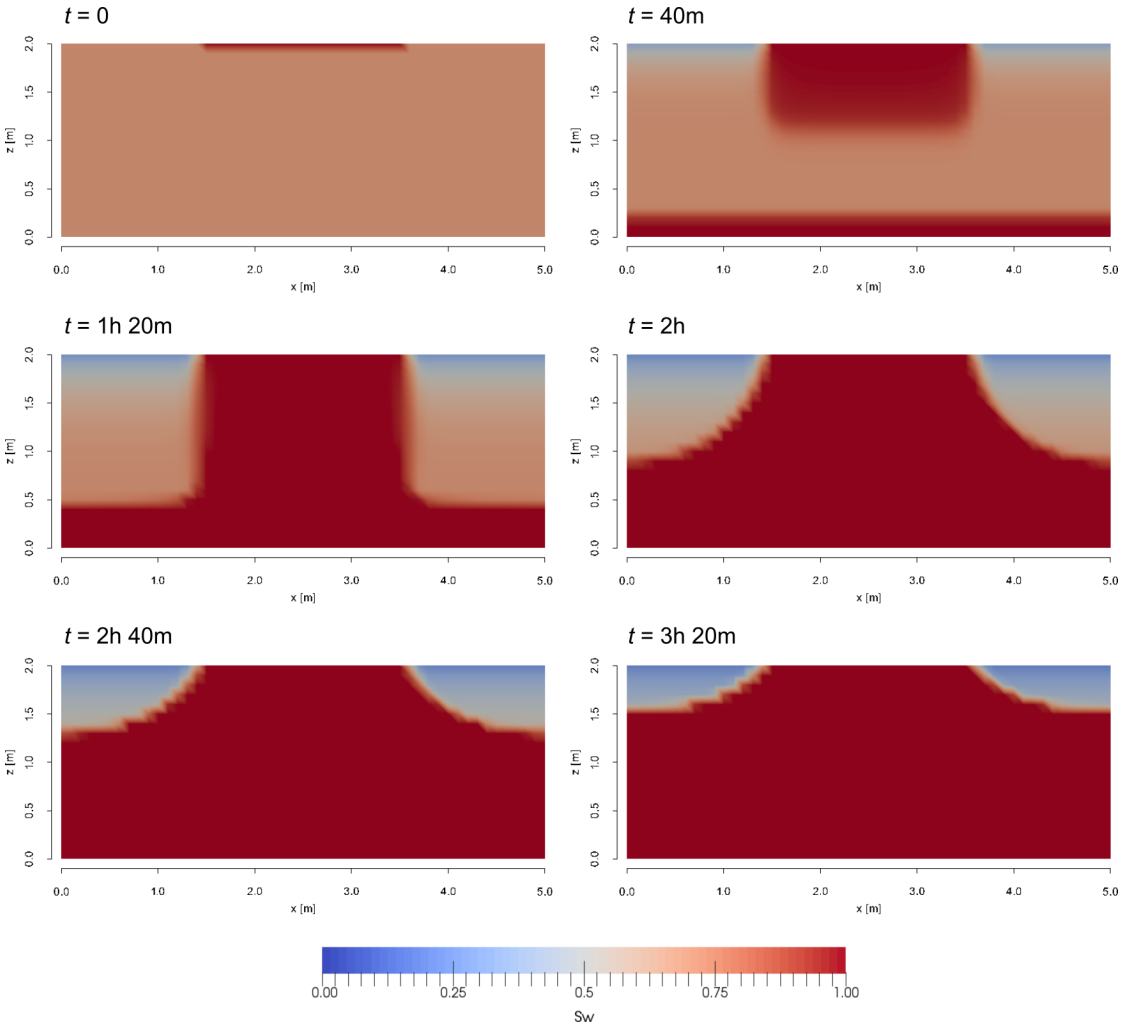

Figure 5.16: Saturation in the subsurface flow model of the coupled model for set-up 2 (wet)

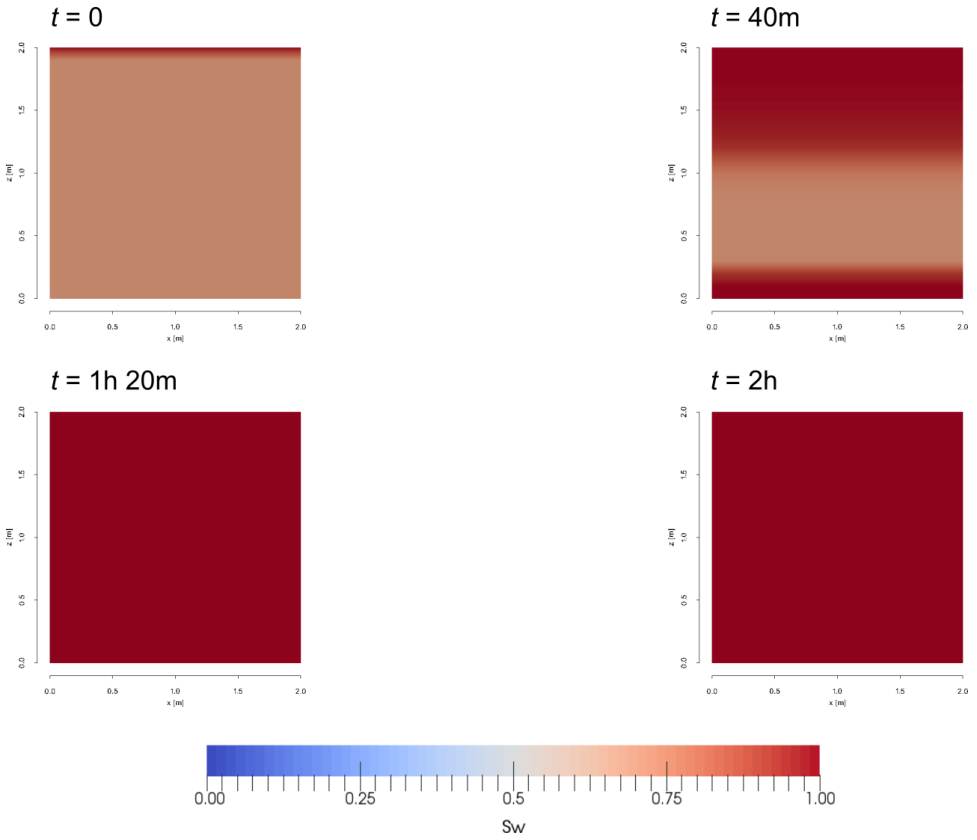

Figure 5.17: Saturation in the subsurface flow model of the coupled model for set-up 3 (wet)

Set-up	**DuMu$^{\mathbf{X}}$** [kg]	**HMS** [kg]
Set-up 1 (dry)	3212.790	3212.800
Set-up 1 (wet)	1170.000	1170.000
Set-up 2 (dry)	1385.180	1385.190
Set-up 2 (wet)	950.943	950.944
Set-up 3 (dry)	1285.120	1285.120
Set-up 3 (wet)	468.000	468.001
Set-up 4 (dry)	3216.410	3216.440
Set-up 4 (wet)	1185.000	1185.010
Set-up 5 (dry)	1358.440	1358.450
Set-up 5 (wet)	943.272	943.274
Set-up 6 (dry)	1286.570	1286.570
Set-up 6 (wet)	474.000	474.003

Table 5.3: Exchange fluxes for the subsurface- and surface flow models for the coupled simulation models

Mass balance

The mass balance for the exchange flux in the coupled simulation was checked for all the above-mentioned set-ups. This was done by measuring the flux through the coupling interface for both DuMu$^{\mathrm{X}}$ and HMS and comparing these values. The results of the mass balance checks are given in table 5.3. As can be seen from these results, the exchange flux is always either exactly the same for the two models or is within the range of the computational error. Thus for the tested set-ups, no mass is lost due to the tensor operations such as interpolation, etc. during the coupling process.

Simulation time

Since network communication is much slower than local memory access (Lee Hutchinson, 2012) and since communication between the coupled models in the set-ups demonstrated here takes place over a network connection because of the use of the

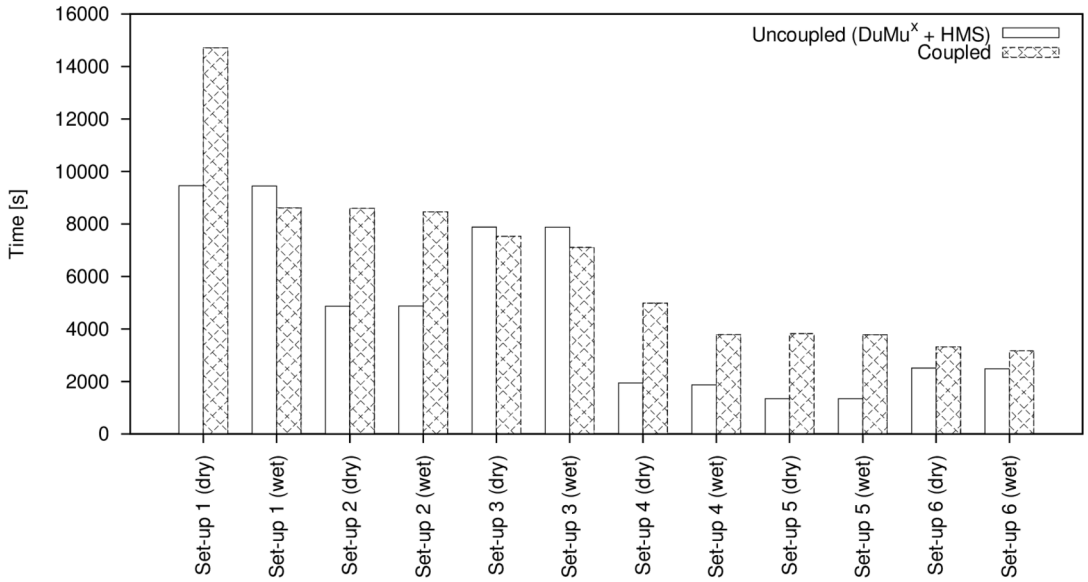

Figure 5.18: Average simulation times for 10 runs of individual uncoupled models and the coupled model.

XML-RPC protocol, it is expected that a coupled simulation would run slower than uncoupled simulations. In order to measure the overhead associated with the coupling of models, for each set-up mentioned in the previous sections, 10 simulation runs were performed for:

- the uncoupled DuMu^{X} model
- the uncoupled HMS model
- the coupled DuMu^{X} and HMS model

It should be noted that:

- because of the coupled nature of the problem, the exchange fluxes for the uncoupled problems need to be assumed, and it isn't entirely possible to break-up the time into three clear-cut parts (one each for the coupled models and one for the tensor exchange server).
- the Java Virtual Machine performs Just in Time (JIT) compilations and other optimisations (Lindholm et al., 2013). As a result, the performance of Java

Set-up	DuMu^{X} uncoupled	HMS uncoupled	Coupled	% change
Set-up 1 (dry)	38.71	9424.50	14 711.99	55.47
Set-up 1 (wet)	23.85	9424.49	8613.49	−8.84
Set-up 2 (dry)	38.86	4833.53	8601.31	76.53
Set-up 2 (wet)	42.87	4833.53	8467.44	73.64
Set-up 3 (dry)	15.68	7865.27	7532.70	−4.42
Set-up 3 (wet)	9.37	7865.27	7112.61	−9.68
Set-up 4 (dry)	165.90	1777.76	4982.87	156.37
Set-up 4 (wet)	100.05	1777.76	3790.72	101.87
Set-up 5 (dry)	189.83	1153.28	3822.86	184.63
Set-up 5 (wet)	193.37	1153.28	3780.34	180.72
Set-up 6 (dry)	66.40	2447.89	3322.56	32.15
Set-up 6 (wet)	39.20	2447.89	3174.47	27.64

Table 5.4: Average run-times for different test set-ups averaged over 10 simulation runs. Simulated time period = 12 500 s. The % change is calculated as [(coupled - (DuMu^{X} uncoupled + HMS uncoupled)) × 100] / coupled

applications is difficult to measure and will vary both with time and with the platform on which the application is run (Goetz, 2004). Moreover, since the code paths for the coupled and uncoupled models are different, the possibility that optimisations applied for the coupling code in the coupled HMS set-up also affect the rest of the model cannot be entirely ruled out.

Due to these reasons, these results are intended to give a general trend of the performance of the coupled model and they aren't claimed to be very accurate.

For the uncoupled models used in the tests, a constant value for the sink term was set for HMS while a Dirichlet boundary condition was used for DuMu^{X} . The tests were carried out with both models running on a single computer with a 64 bit Intel Core i5 450M processor with 2 cores and support for 4 threads. The operating system used was 64 bit Debian Wheezy version 3.2.57-3+deb7u1 with the Linux kernel version 3.2.0-4-amd (Debian Project, 2015). DuMu^{X} was compiled using the GNU Compiler Collection (gcc) version 4.7.2 with the -O3 optimisation flag (GNU

Project, 2015) without parallelisation support. HMS and the tensor exchange server were run on the Oracle HotSpot Java Virtual Machine for servers (build 25.5-b02) running Java version 1.8. Both these Java applications were run with parallelisation support. In order to avoid differences due to the use of different programming languages, the `time` command available on UNIX-like systems (Wikipedia, 2015n) was used to measure the real time used for each simulation.

The average times for each of the simulation runs (simulated time period = $12\,500\,\mathrm{s}$) is given in table 5.4 while figure 5.18 shows a histogram produced using these results. From these results it is clear to note that:

- the DuMu$^{\mathrm{X}}$ simulations are generally much faster than HMS (see table 5.4) for these set-ups. This is because the time step size used by DuMu$^{\mathrm{X}}$ are much larger than those for HMS. While the time steps used by HMS range from fractions of a second to a couple of seconds, the maximum time step size for DuMu$^{\mathrm{X}}$ for these set-ups has been limited to $30\,\mathrm{s}$.

- as expected, the coupled simulations are generally slower than the sum of the times for the individual uncoupled simulations. As explained earlier, this is due to the overhead for transferring the values over the network and for the tensor operations. However, the largest observed change is approximately $185\,\%$, i.e. the coupled simulation runs a little less than three times slower than the uncoupled simulations totalled.

- For three of the set-ups, the uncoupled HMS simulation is actually slower than the coupled simulation. While this might seem counter-intuitive at first, this behaviour is easy to explain by taking the differences in the coupled and uncoupled HMS models into account. While the sink term in HMS representing the exchange flux varies with time in the coupled simulation, it is set to a constant value in the uncoupled set-up. As a result, the modelled conditions vary in the two models and in these three particular cases, HMS is able to use larger time step sizes in the coupled set-up resulting in a faster simulation run.

To summarise these results, it should be reiterated that due to the coupled nature of the problem at hand, it is difficult to compare the performance of uncoupled set-ups with coupled ones. The coupled set-ups should generally perform slower than the uncoupled set-ups but depending on the case being simulated, the models might be able to gain performance by choosing larger time step sizes.

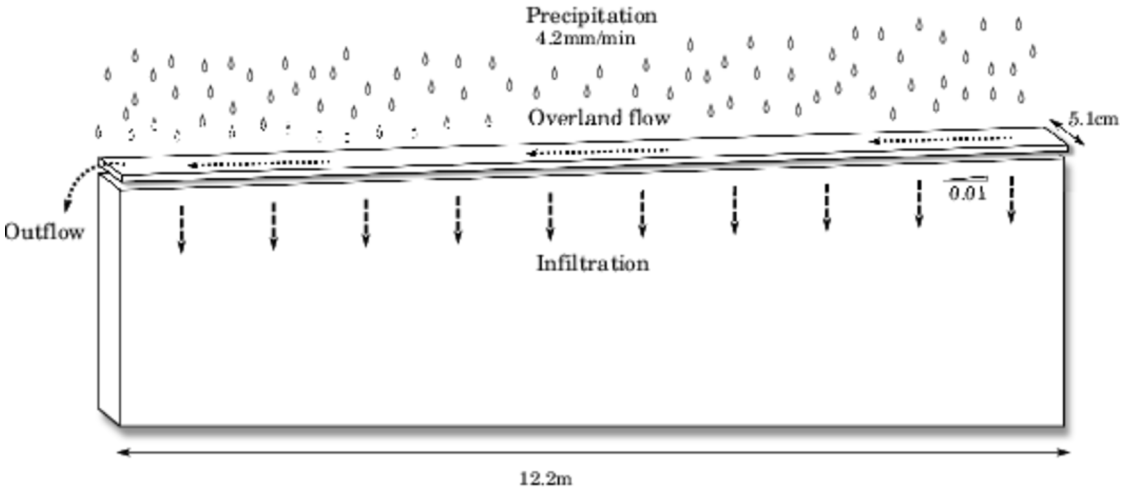

Figure 5.19: Laboratory set-up of the experiment of Smith and Woolhiser (1971)

5.3.5 Smith and Woolhiser experiment

Analytical solutions and test cases for problems describing Hortonian runoff (Cea et al., 2014, Stephenson and Meadows, 1986, Tatard et al., 2008), exist and have been used in previous studies (Cea et al., 2014, Gottardi and Venutelli, 1993, Kollet and Maxwell, 2006, Simons et al., 2014). Similarly, analytical solutions for subsurface flow in domains with inclined surfaces are also available (Troch et al., 2002, Verhoest and Troch, 2000). However, as Kolditz et al. (2008) have noted, '[d]espite abundant test cases for flow in saturated and unsaturated porous media [...] or overland flow [...] only a few examples are available for testing numerical models for coupled surface-subsurface systems.'

An important experiment that has been used to validate multiple studies dealing with coupled surface and subsurface flow (e.g. (Kolditz et al., 2008, Singh and Bhallamudi, 1998, Smith and Woolhiser, 1971)) is the Smith and Woolhiser experiment (Smith and Woolhiser, 1971). This experiment was chosen to validate a coupled surface-subsurface flow for this work because of availability of data for both the surface and subsurface flow.

Model set-up

Figure 5.19 shows a schematic diagram of the set-up used by Smith and Woolhiser for their laboratory experiment. It consisted of a flume of soil measuring $12.2\mathrm{m}$ in

length and $5.1\mathrm{cm}$ in width, and having a slope of 0.01. The soil used for the experiment consisted of locally obtained river-deposited sand. Although soil bulk density measurements showed variations along both the horizontal and vertical directions, subsequent studies have shown that the lateral flow effects in the soil were only marginal (Delfs et al., 2009). An artificial rainfall of $4.2\mathrm{mm/min}$, which was 2–3 times the saturated soil conductivity, was simulated for $15\mathrm{min}$. A light oil resembling kerosene was used for this purpose instead of water to prevent algal growth. The oil had a density of $756\mathrm{kg/m^3}$ and a kinematic viscosity of $2.83 \times 10^{-6}\,\mathrm{m^2\,s^{-1}}$, which is almost double that of water. Experimental data for the soil saturation was obtained at different depths along a vertical profile along the middle of the flume. In the overland domain, seepage was allowed at both ends of the flume and the seepage at the downstream end of the flume was collected and measured (Delfs et al., 2009, Smith and Woolhiser, 1971).

For reproducing the results of the laboratory experiments by Smith and Woolhiser a coupled surface and subsurface flow model has been used. The coupled model set-up is similar to that used for the test cases from the previous section. Overland flow is simulated using a two-dimensional set-up in HMS. The subsurface flow is simulated using a two-dimensional model set-up in DuMu$^{\mathrm{X}}$. The spatial and temporal discretisations for both models were different, something that, to the knowledge of the author, has not been attempted before. The overland flow model consisted of 6100 computational cells with a cell size of $1\,\mathrm{mm} \times 1\,\mathrm{mm}$. Since HMS uses a cell-centred finite volume scheme, the number of computational nodes this is also the same. For the subsurface flow model, a total depth of $0.7\,\mathrm{m}$ was simulated with a cell size of $2\,\mathrm{cm} \times 1\,\mathrm{cm}$ resulting in $42\,700$ computational cells and $43\,381$ computational nodes. The coupling interface is one-dimensional with different space ($1\,\mathrm{mm}$ and $2\,\mathrm{mm}$ for HMS and DuMu$^{\mathrm{X}}$ respectively) and time discretisations (adaptive time steps for both models) for the overland and subsurface flow models. The physical parameters used for the two models are summarised in table 5.5. Explanations of the three bottom friction coefficients and the immobile water depth can be found in Delfs et al. (2009).

Results

Figures 5.20 and 5.21 show the results obtained for the coupled simulation model for the outflow at the downstream end of the flume and for the saturation front in the subsurface respectively. It can be seen from these figures that the obtained results

Parameter	**Value**
Fluid kinematic viscosity	$2.95 \times 10^{-6}\,\mathrm{m^2\,s^{-1}}$
Fluid density	$756\,\mathrm{kg\,m^{-3}}$
Bottom friction coefficient, C'	$1.33 \times 10^{6}\,\mathrm{m^{-1}\,s^{-1}}$
Bottom friction coefficient, j	1
Bottom friction coefficient, l	2
Soil porosity	0.42
Soil residual saturation	0.05
Soil hydraulic conductivity	$2.83 \times 10^{-5}\,\mathrm{m\,s^{-1}}$
Immobile depth	$0.001\,\mathrm{m}$

Table 5.5: Modelling parameters for the Smith and Woolhiser experiment (Delfs et al., 2009)

Van Genuchten α [$\mathrm{Pa^{-1}}$]	**Van Genuchten** m [-]	**RMSE** [$\mathrm{mm\,min^{-1}}$]
0.0050	0.54	0.220
0.0050	0.56	0.153
0.0050	0.60	0.189
0.0060	0.65	0.203
0.0065	0.70	0.191
0.0070	0.75	0.177

Table 5.6: Root Mean Square Error for the coupled simulation set-ups

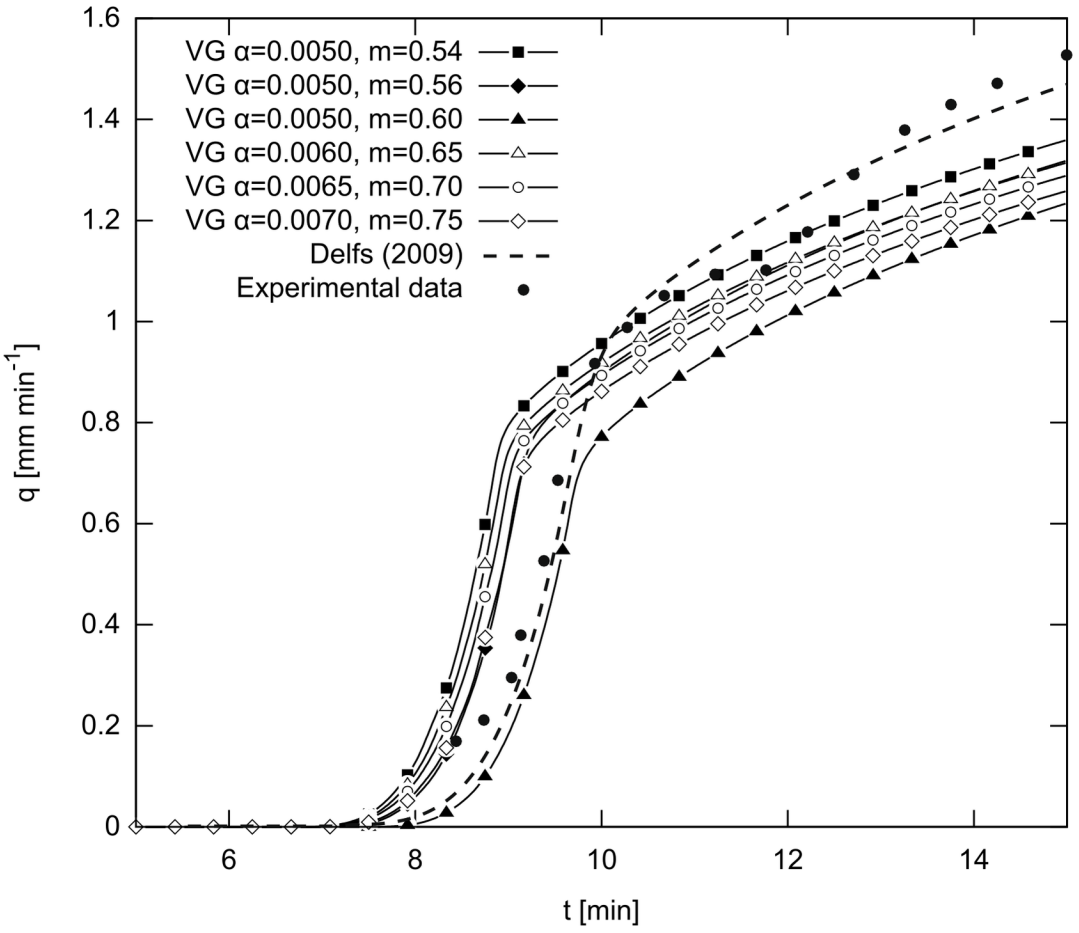

Figure 5.20: Comparison of discharge at the downstream end of the flume for the coupled simulations using tensor objects, Delfs (2009) and experimental results by Smith and Woolhiser (1971). Values for the Van Genuchten parameter α are in Pa^{-1}.

are in a similar range to the measurements made by Smith and Woolhiser (1971) but the results from the coupled model vary slightly from those observations from the laboratory model. From figure 5.20 it can be seen that there was no discharge at the outflow for approximately the first $8\,\mathrm{min}$. After this time, there is a sharp increase in the measured discharge and around $10\,\mathrm{min}$ from the start of the artificial rainfall event, the rate of increase of discharge falls rapidly. Although the coupled model is able to reproduce these three stages, also reproduced by Delfs et al. (2009) the simulated discharges by the coupled DuMu$^{\mathrm{X}}$ and HMX model are always much lower than those measured by Smith and Woolhiser (1971), especially after $12\,\mathrm{min}$,

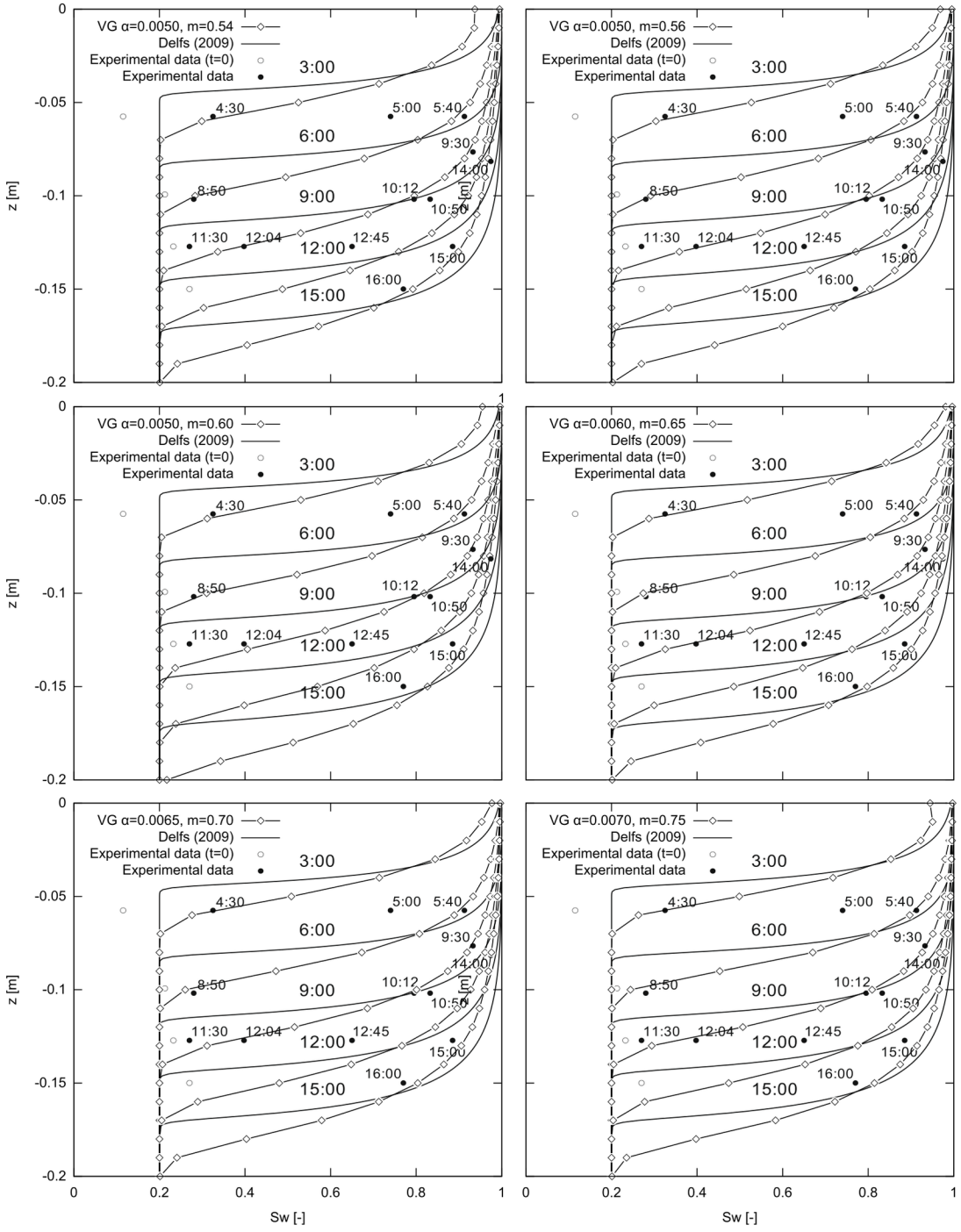

Figure 5.21: Comparison of moisture front in the subsurface for the coupled simulations using tensor objects, Delfs (2009) and experimental results by Smith and Woolhiser (1971). Values for the Van Genuchten parameter α are in Pa^{-1}.

and none of the simulations is able to reproduce the results perfectly. This implies that the volume of the infiltrating fluid is much larger in case of the coupled model than observed in the laboratory or by other models e.g. by Delfs et al. (2009). This trend is also reflected in the movement of the moisture front in the subsurface model (see figure 5.21). Here it can be seen that moisture front moves at a much faster rate than observed in the laboratory. As a result, lower soil layers in the coupled model become saturated much earlier than in the laboratory experiment. The reasons for this are not completely understood. The approach used by Delfs et al. (2009) is based on hydraulic compartments where subsurface flow, surface flow and the interface between the two are modelled separately and the compartments interact via exchange fluxes. It is supposed that further incorporating elements of this approach, especially the thin hydraulic interface layer between the subsurface- and surface flow models might improve the results of the coupled DuMu$^{\text{X}}$ and HMS simulation.

In order to try and improve the simulation results, an attempt was made to vary the Van Genuchten parameters α and m (see section 5.1.3) to try and calibrate the simulation model using the results of the laboratory model. The best results obtained from these simulation are summarised in table 5.6 and are also reflected in figures 5.20 and 5.21.

Conclusions

The subsurface flow model DuMu$^{\text{X}}$ has been successfully coupled with the surface flow model HMS for the modelling of runoff and infiltration processes. Using the tensor exchange server, it is possible to couple models written in different languages (C++ and Java). Moreover, with the use of tensor objects, it is possible to automatically adapt information to suit the requirements of the coupled models through operations such as transformation of physical state variables, mapping, interpolation in space and time, etc. However, the obtained results need to be improved and for doing this, it is necessary to improve the way the infiltration processes are simulated by the coupled model. One possible method of achieving this is to use an infiltration model for calculating the exchange flux at the interface between the subsurface and surface flow models (see Haverkamp et al. (1988), Mirzaee et al. (2014) for a comparison of some of the available infiltration models). This task is however out of the scope of the current work, which only seeks to demonstrate the utility of tensor objects in the coupling of hydroinformatic models and that of the TES for adapting the information to the requirements of coupled models through operations

such as mapping, interpolation, etc. This goal has been successfully achieved in this application example.

5.3.6 Summary

In the previous sections, coupling of the subsurface flow model DuMu^{X} with the surface flow model HMS has been demonstrated. Through the use of tensor objects, the coupled model is able to handle the adaptation of information between the formats used by the two models through tensor operations involving transformation of physical state variables, interpolation in time and space, etc. These operations have been demonstrated to be mass conservative, since within the range of computational error, the exchange flux for coupled models is the same. The coupling of the models has also been demonstrated by reproducing the results of a well-known laboratory experiment. Although the results of this simulation need to be improved, they still successfully demonstrate the utility of using tensor objects and the TES for coupling.

5.4 Information management system for Rhine section

The final application example demonstrates the use of tensor objects in an information management system for a section of Rhine. For hydroinformatics related projects, an information management system typically provides the following functionality (see figure 5.22):

- **Store**: standardised formats for storing information from various sources e.g. measurement data, simulation results, maps, reports, etc.
- **Search**: a means for searching for data based on user-defined criteria. This is typically done with the help of metadata (see section 2.1), which can be created at the time of storing data.
- **Retrieve**: retrieve the stored data for use with other models
- **Edit**: make modifications to the existing data or update it with newer versions
- **Analyse**: provide simple analysis of the data e.g. simple statistics, etc.

HydroDesktop (Ames et al., 2012) and WISKI (KISTERS, 2015) are two examples of existing information management services for hydrological data.

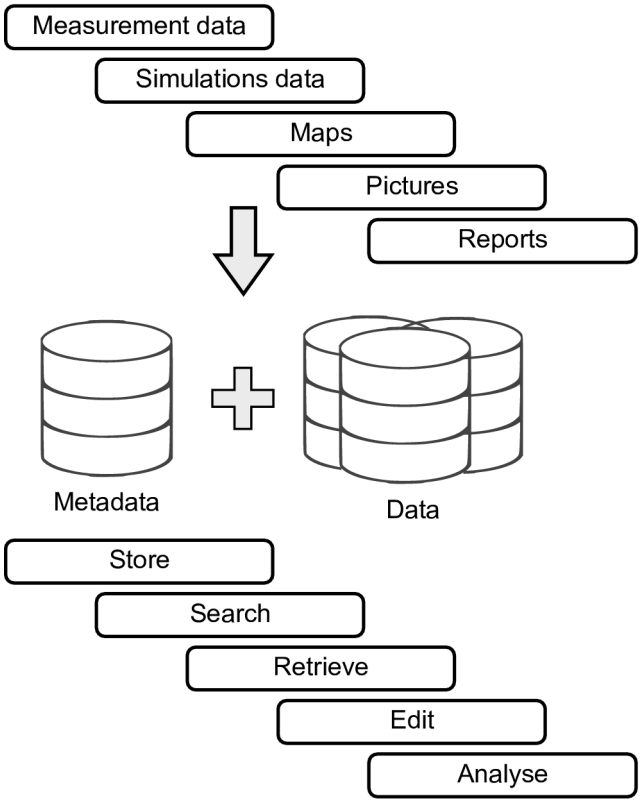

Figure 5.22: Information management system concept

For this application example, an information management system has been set up for a virtual project dealing with two-dimensional hydrodynamic simulation of a section of the river Rhine. It provides a means of storing information related to the project and for viewing these results through a WebGIS interface in the browser in the form of maps, graphs, tables, etc. Thus the set-up used in this application example is more complex than those in the previous sections and consists of multiple hydroinformatic models including a simulation model, a database model, a GIS model, etc. coupled together within the information management system. It should be noted that the primary purpose of this application example is to demonstrate the utility of tensor objects in an information management application for hydroinformatics related projects. As a result, the imperfections in the hydrodynamic modelling were considered of minor importance.

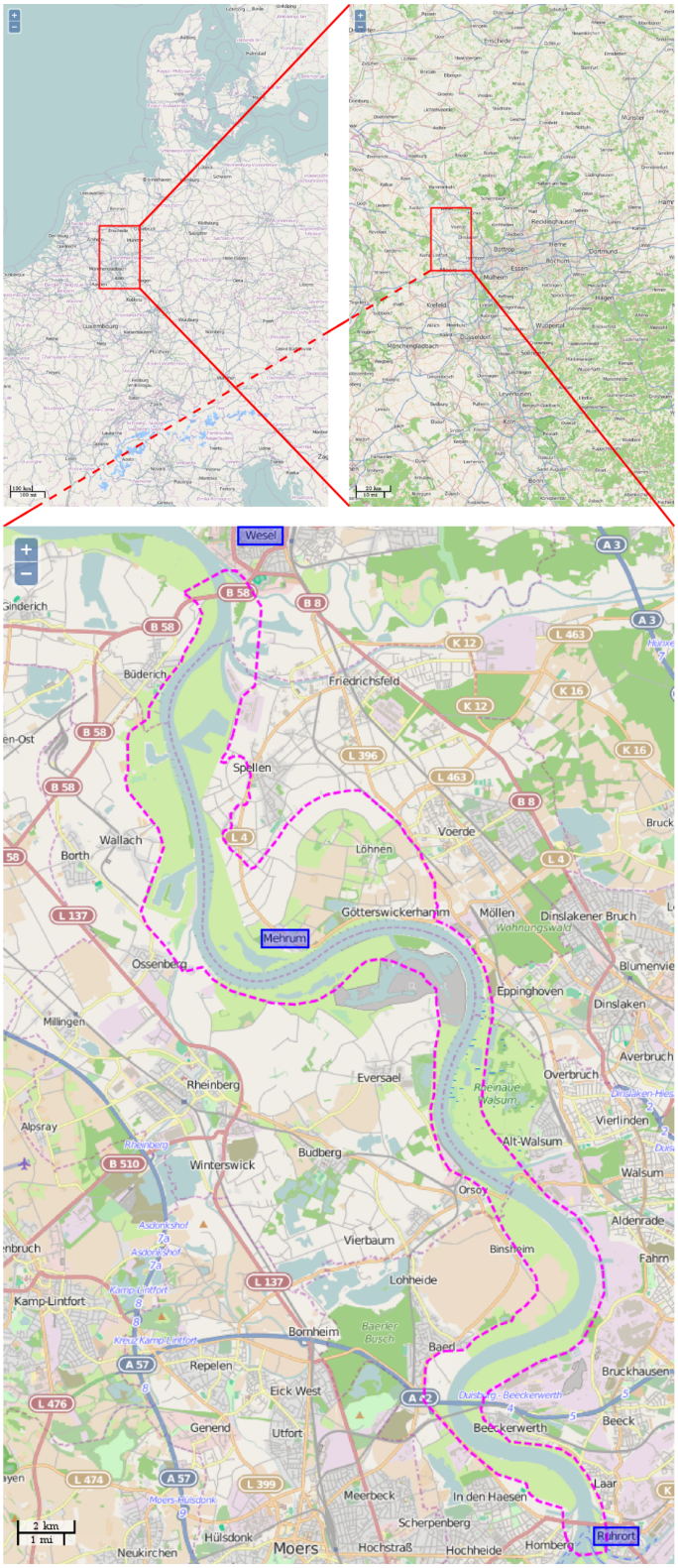

Figure 5.23: Modelled section of the river Rhine between Ruhrort and Wesel

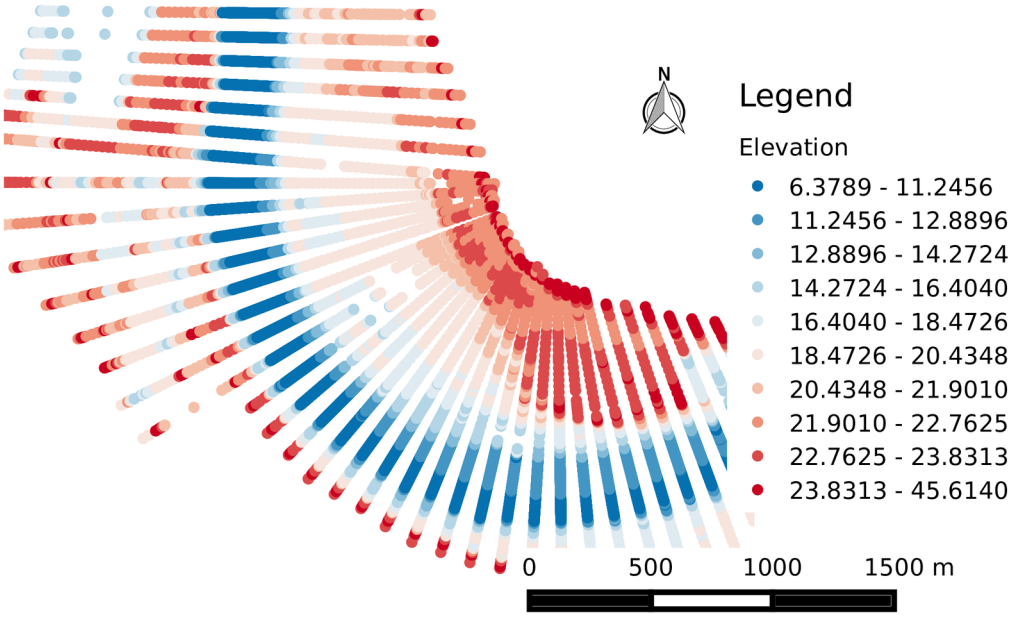

Figure 5.24: River cross sections. (Elevations in m above NN.)

5.4.1 Study area

The study area for the hydrodynamic model lies in the lower Rhine region in the northwest of Germany. The modelled area is approximately $50\,\mathrm{km}^2$ in area, consisting of a $34.4\,\mathrm{km}$ long section of the Rhine between the tributaries Ruhr and Lippe and a polder near Mehrum (see figure 5.23). The polder is used mainly for agricultural purposes and is protected against flooding through dykes (Chua et al., 2001).

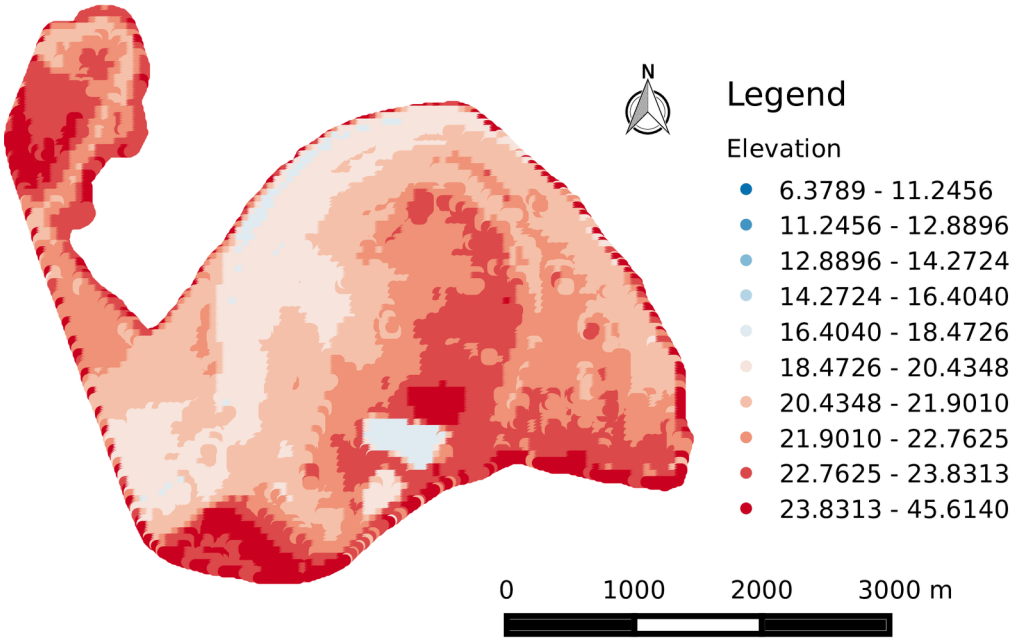

Figure 5.25: Elevation data in m above NN for the Polder

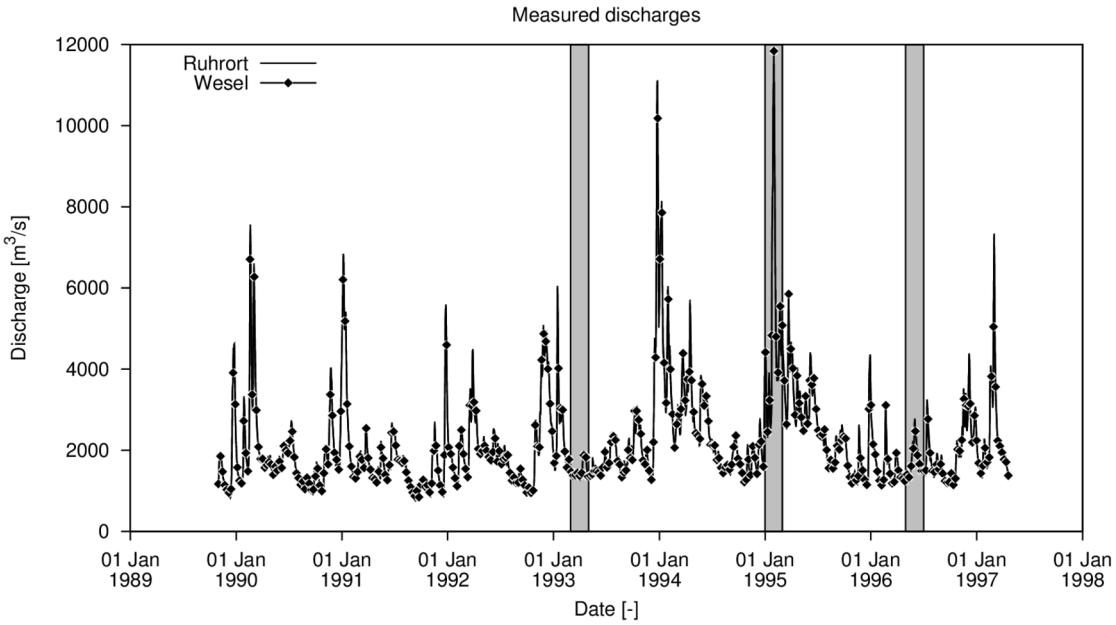

Figure 5.26: Measured discharges. Highlighted sections are time windows used for calibration (left), validation (right) and the flood event (centre).

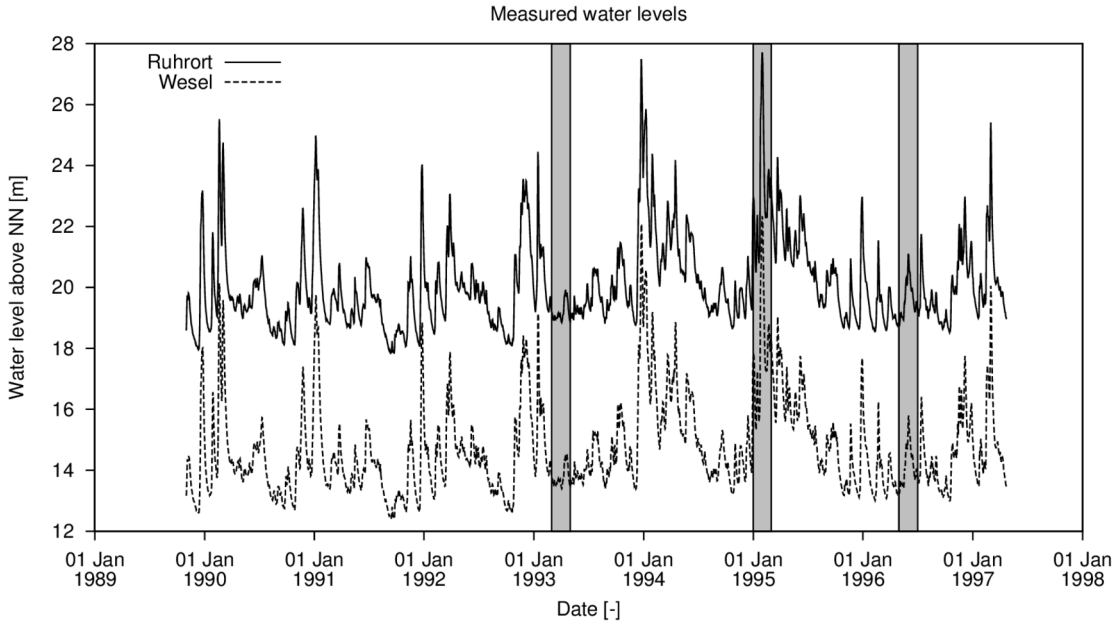

Figure 5.27: Measured water levels. Highlighted sections are time windows used for calibration (left), validation (right) and the flood event (centre).

5.4.2 Available data

The following data were available for setting up the model:

- **Measured cross-sections** every $100\,\mathrm{m}$ along the river (see figure 5.24)
- **Digital elevation model** with a resolution of $100\,\mathrm{m}$ for the polder (see figure 5.25)
- **Discharges** measured every hour at the measurement stations located at Ruhrort and Wesel (see figure 5.26)
- **Water levels** above NN (WSV, 2015a) measured every hour at the measurement stations located at Ruhrort and Wesel (see figure 5.27)

5.4.3 Hydrodynamic model set-up

TELEMAC-2D (open TELEMAC-MASCARET, 2015) is the model that has been used for the hydrodynamic simulation of a $34.4\,\mathrm{km}$ long section of the Rhine river along with a polder. A computational mesh (see figure 5.28) for a 2D hydrodynamic model is set up in TELEMAC-2D using the above data. The mesh elements are refined along the river channel and the dyke separating the polder from the river and the floodplains. It consists of 13127 nodes and 25761 elements. Elevations at the vertices of the mesh are interpolated from the measured cross-section data and the digital elevation model (see section 5.4.2) using inverse distance interpolation. Separate values of the Manning roughness coefficient were used for the river channel and the polder/floodplains. As the upstream boundary condition was set as the measured discharge time series at Ruhrort. The measured water levels at Wesel were used as the downstream boundary conditions.

5.4.4 Calibration

The model was calibrated by adjusting the Manning roughness coefficient for the main river channel. For this purpose, the time window from 1st March 1993 to 1st May 1993 was chosen (see figures 5.26 and 5.27) because the flow during this time period was close to the average flow conditions over more than 10 year during the time period for which the data were available. The measured water levels at Ruhrort were used as the calibration parameter. A value of $0.020\,\mathrm{m}^{-1/3}\mathrm{s}$ for the Manning roughness coefficient for the main channel was found to give satisfactory

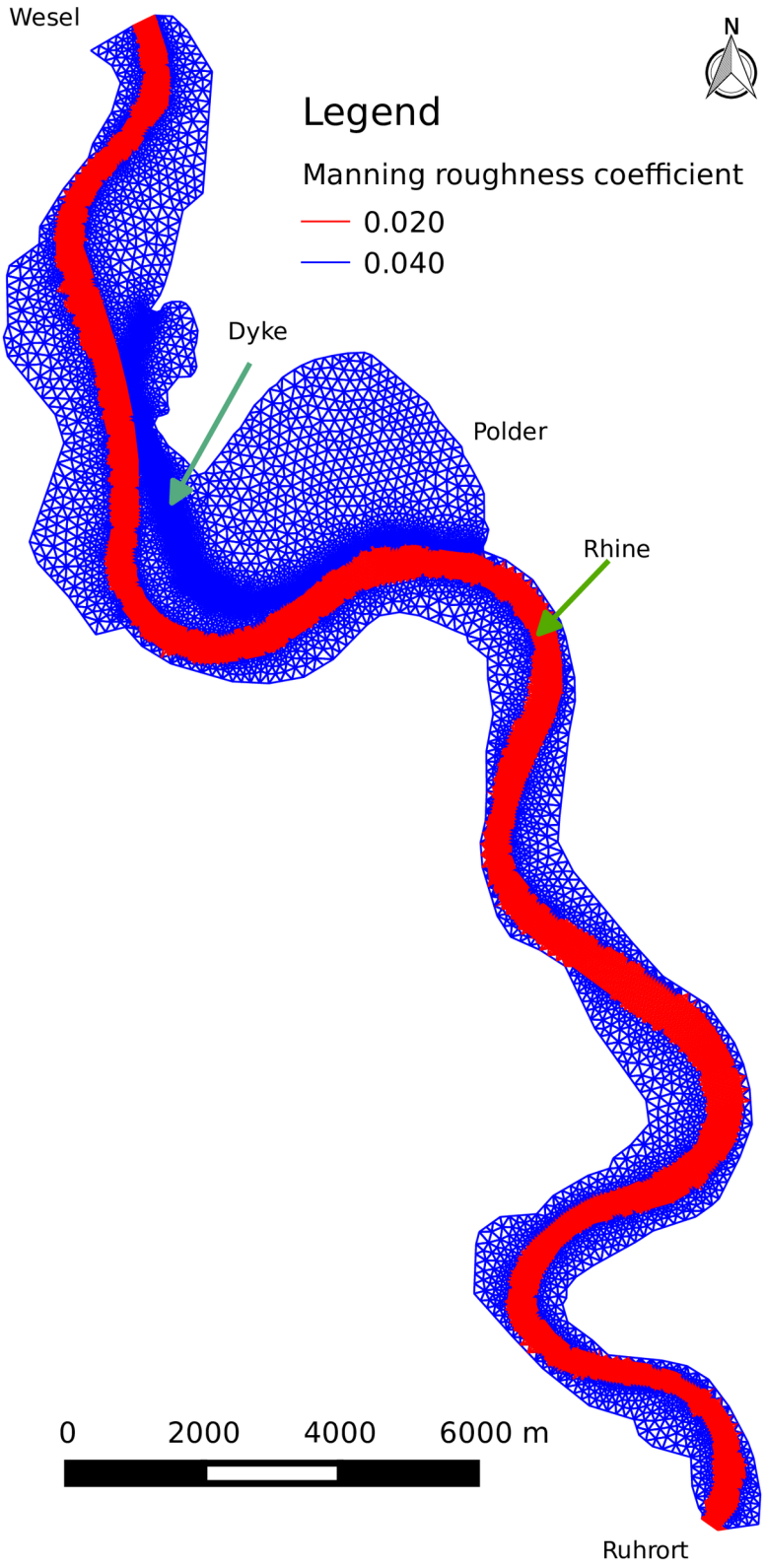

Figure 5.28: Computational mesh for hydrodynamic model, also demonstrating the different values of the Manning roughness coefficient ($\mathrm{m}^{-1/3}\mathrm{s}$) for the main river channel and for the floodplains and polder.

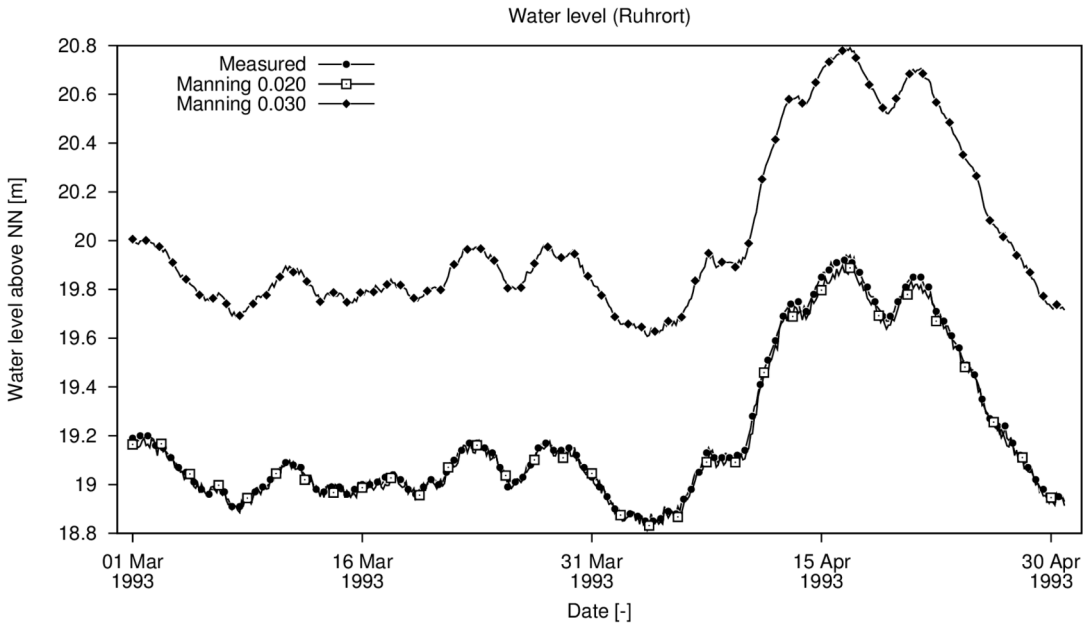

Figure 5.29: Calibration of water levels at Ruhrort

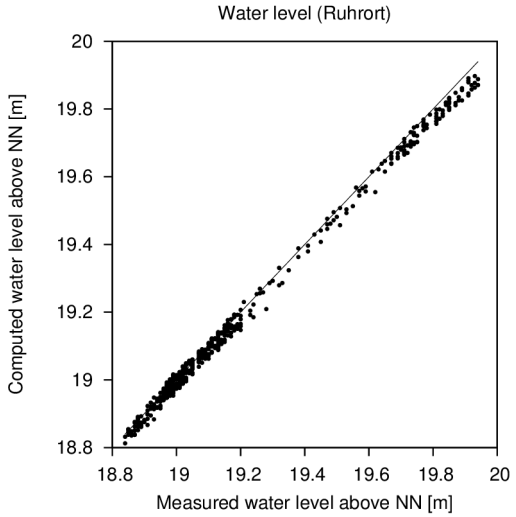

Figure 5.30: Comparison of measured and computed (simulation period=01 Mar 1993 – 01 May 1993, Manning=$0.020\,\mathrm{m}^{-1/3}\mathrm{s}$) water levels at Ruhrort. Solid diagonal line indicates equal values.

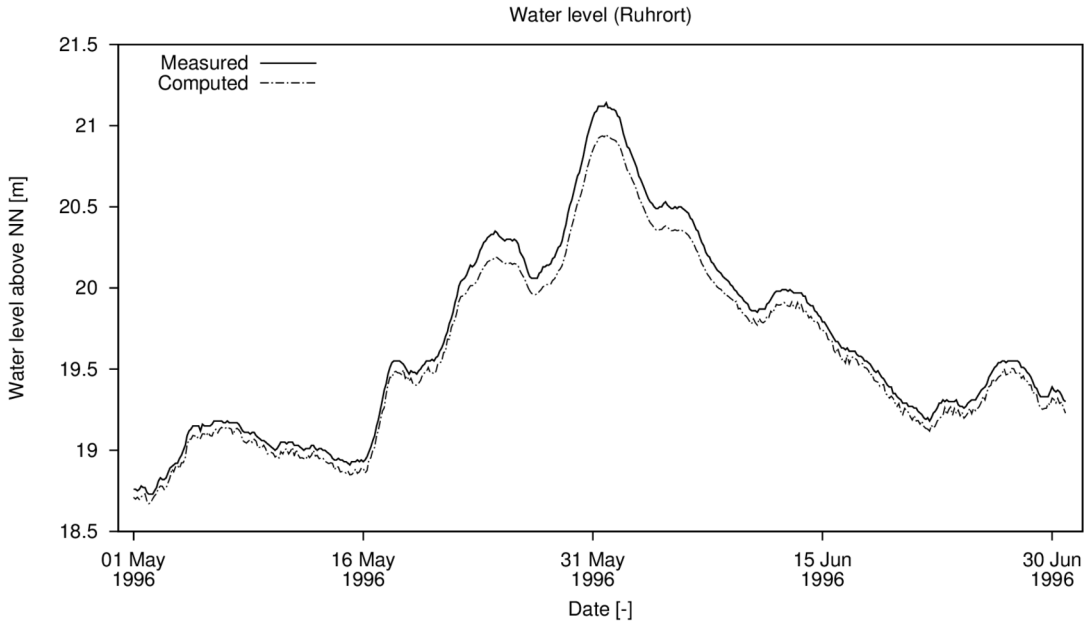

Figure 5.31: Validation of water levels (Manning=$0.020\,\mathrm{m}^{-1/3}\mathrm{s}$) at Ruhrort

results (see figures 5.29 and 5.30) with a Root Mean Square Error (RMSE) Value of $0.025\,\mathrm{m}$ and a Pearson Correlation Coefficient of 0.998. As a comparison, the results obtained using a Manning's roughness coefficient of $0.030\,\mathrm{m}^{-1/3}\mathrm{s}$ for the main channel also show good correlation to the observed values (Pearson Correlation Coefficient of 0.996) but a much higher RMSE ($0.81\,\mathrm{m}$). Good correlation in this case means that the measured and computed values show good covariance, as seen from the value of the Pearson Correlation Coefficient being very close to 1.0, but do not overlap, as seen from the high value of the RMSE. This is the reason why the curves of the measured values and the values computed using a value of $0.030\,\mathrm{m}^{-1/3}\mathrm{s}$ for Manning's roughness coefficient are roughly parallel to each other but don't overlap (see Figure 5.29).

5.4.5 Validation

After calibrating the model, validation was carried out using data from the time period from 1st May 1996 to 1 August 1996 (see figures 5.26 and 5.27) (see figures 5.31 and 5.32). The validation results are found to be less accurate than the calibration results (see figure 5.31). Compared to an RMSE value of $0.025\,\mathrm{m}$ for calibration,

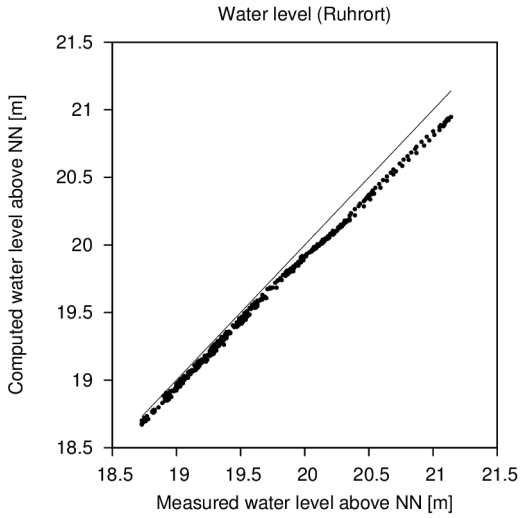

Figure 5.32: Comparison measured and computed (simulation period=01 May 1996 – 01 Jul 1996, Manning=0.020) water levels at Ruhrort. Solid diagonal line indicates equal values.

the validation results have an RMSE value of $0.087\,\mathrm{m}$ for the measured water levels at Ruhrort using a value of 0.020 as the Manning roughness coefficient for the main river channel. It is also plain to see that although the model results show a high degree of correlation (Pearson correlation coefficient = 0.999), the model consistently underestimates the water levels at Ruhrort, especially for higher values of water levels (see figure 5.32). Comparing figures 5.30 and 5.32 it can be concluded that similar to the calibration results, the results for the validation are satisfactory for waterlevels up to $20\,\mathrm{m}$ above NN and the model needs to be further calibrated for higher water levels. However, since the purpose of this application is to demonstrate the functionality of tensor objects in an information management application, and because the validation results still have a low RMSE value, these imperfections in the validation results were ignored and no attempt was made to improve them.

5.4.6 Information management system

Simulation results obtained from the TELEMAC-2D simulations are in the form of the SELAFIN (TELEMAC-MASCARET, 2014) files produced by TELEMAC. For the purpose of storing them in the information system, the simulation results are extracted from the SELAFIN files using Python (pytel) (Foundation, 2015b, open

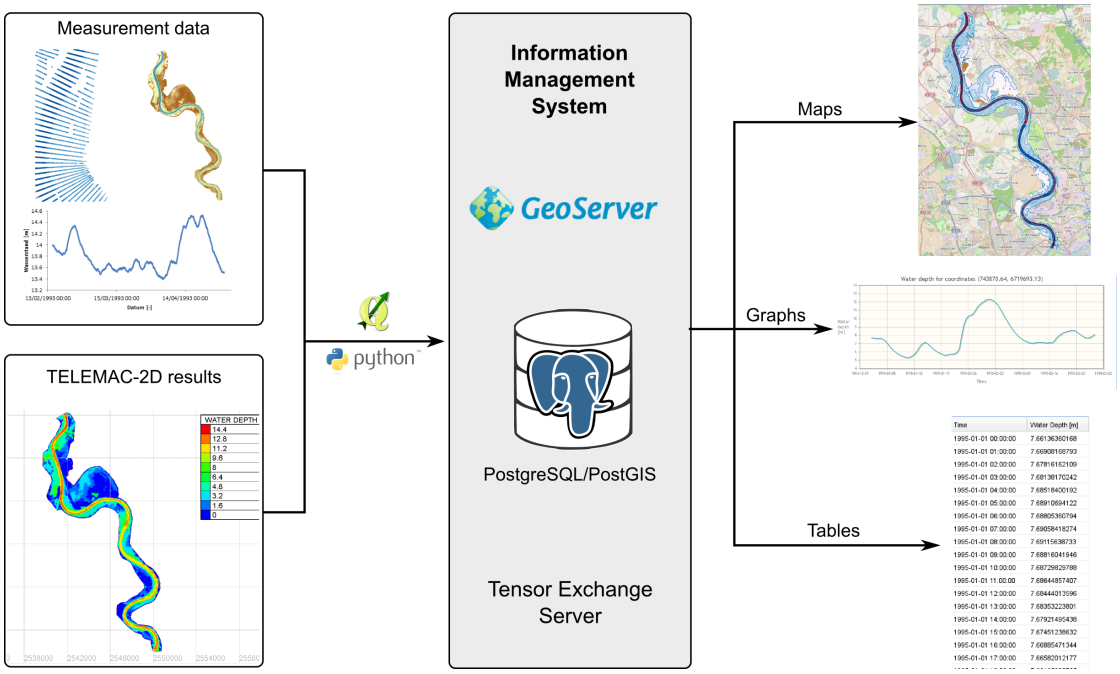

Figure 5.33: Information management system for Rhine simulation data.

TELEMAC-MASCARET, 2015) scripts, Fudaa-prepro (CETMEF, 2015) and Blue Kenue (NRC, 2015) (see figure 5.33). Using these applications the following information was extracted from the results of the hydrodynamic simulation:

- **Geometric** information e.g. bathymetry. This can also be done in the form of raster GIS data for storage using Geoserver.
- **Topological** information such as the computational nodes and elements. This can also be done in the form of vector GIS data and be imported into the PostGIS database.
- **time series** e.g. waterlevel time series at computational nodes, etc. These can then be stored in the PostgreSQL database.
- **Raster** GIS data e.g. maps of the water surface at any instance during the simulation for storage using Geoserver, and
- **Vector** GIS data e.g. isolines of the water surface at any instance in time, boundary of the computational domain, etc. for storage using the PostGIS database.

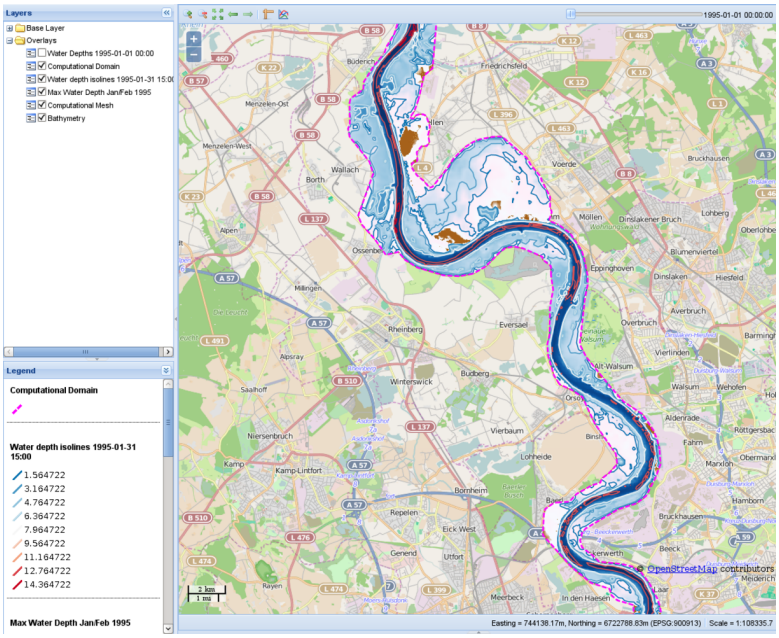

Figure 5.34: WebGIS interface for simulation results

It should be noted that the spatial information was always in the DHDN / 3-degree Gauss-Kruger zone 2 spatial reference system (Butler et al., 2015).

The information management system itself consists of the following components (see figure 5.33):

- **PostgreSQL** database for storing information such as time series data
- **PostGIS** extensions for the PostgreSQL database for storing vector GIS data such as the river cross sections, isolines for water levels, etc.
- **Geoserver** instance for storing raster GIS data and for providing an interface for accessing the raster GIS data and the vector GIS data in the PostGIS database through web services
- **Tensor exchange server** providing an interface for retrieving simulation results (non-geographic information) from the database, performing interpolation, scaling, etc.

The WebGIS interface (see figure 5.34) provides a means of accessing the information stored in the information management system. The main panel of this

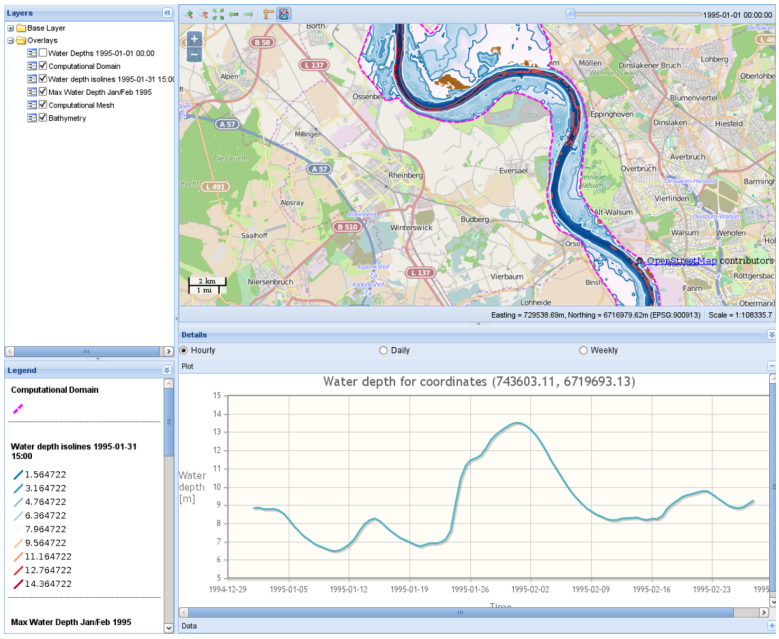

Figure 5.35: Simulation results as a graph. $\Delta t = 1\,\mathrm{h}$

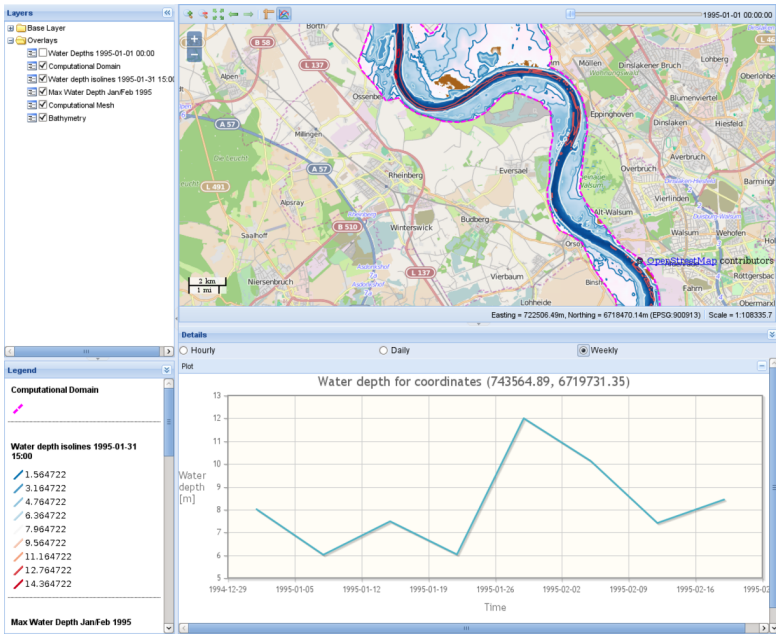

Figure 5.36: Simulation results as a graph. $\Delta t = 1$ week

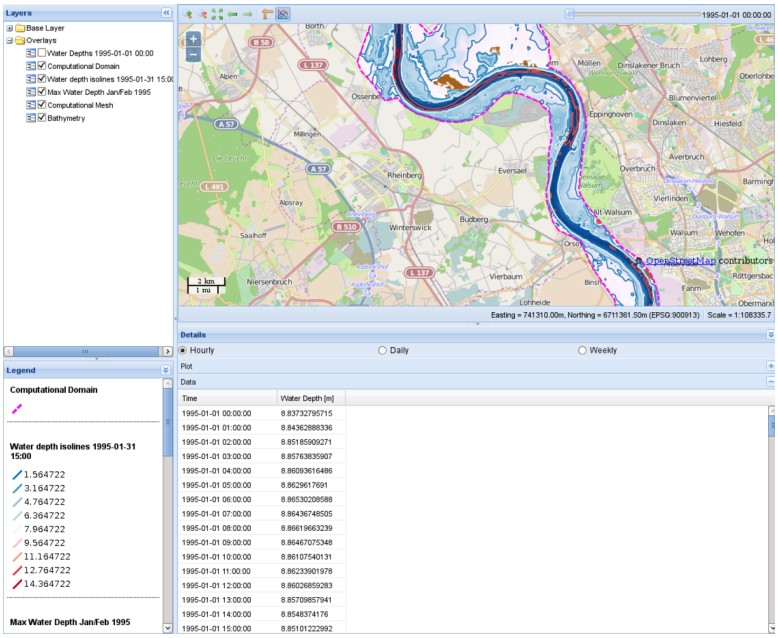

Figure 5.37: Simulation results as a table

interface provides a base map of the world from OpenStreetMap (2015). The raster and vector GIS layers stored in the information management can be displayed on top of this map. Access to this data is provided through Geoserver (Foundation, 2015a) with the help of web services. All the available layers are displayed in the top-left pane with the user having the choice which layers to display and which ones to hide. The legend in the lower-left panel gets automatically updated every time the displayed layers are changed.

It is possible to retrieve simulation results like the waterlevels, profile view of the bathymetry, etc. anywhere inside the computational domain in the form of graphs (see figures 5.35 and 5.36) or tables (see figure 5.37). The retrieval of this information is handled by the Tensor Exchange Server, which is responsible for

- retrieving the data from the PostgreSQL database
- performing interpolation e.g. nearest neighbour interpolation, and
- scaling the data to the requested time interval

The WebGIS interface thus provides a means for interactively retrieving and viewing simulation results using any browser. As mentioned earlier, the current

implementation of the information management system doesn't handle metadata but for larger volumes of data, inclusion of an application such as Geonetwork-opensource (Open Source Geospatial Foundation, 2015) would become indispensable.

5.4.7 Implementation of the WebGIS application

Listing 15 in the appendix shows the HTML and JavaScript code used to implement the interactive WebGIS application. Most of the code is for creating the map, creating the GUI or for providing the interactive features of the WebGIS application, some of which has been excluded in the listing for the sake of clarity. Since our focus is on the use of tensor objects in such an application, the interesting parts of this listing are:

- line 36, which loads the Mimic library for XML-RPC support in JavaScript
- lines 130 to 135, which communicates with the Tensor Exchange Server (TES) and performs the XML-RPC request, and
- line 152, which processes the response from the TES and creates JavaScript objects from this response.

Similar to the previous applications, the TES is implemented using the `CouplingServer`, `CouplingHandler`, `CouplingTensor` and `TensorSet` classes. A description of these classes along with their roles in the coupling process can be found in the sections 5.2.3 and 5.3.3, so they are not discussed here further. However, the implementations of `Operator`s are slightly different in this case because they communicate with the PostgreSQL/PostGIS database through SQL statements and also leverage the basic GIS functionality provided by PostGIS (PostGIS, 2015a) for finding out the intersection point of two lines, computing the distance between two points, performing nearest neighbour interpolation, etc. The implementations of the operators for retrieving time series data from the PostgreSQL database and for creating the cross-sectional profile of the bathymetry are shown in listings 16 and 17 respectively.

5.4.8 Summary

The previous sections have demonstrated one possible implementation of an information management system (see figure 5.22). This implementation is composed of

various different hydroinformatic models and uses tensor objects to provide some of its functionality. It uses a database capable of storing spatially related data to store the results of a two-dimensional hydrodynamic simulation. The WebGIS application is built on top of the information management system and provides a fully-functional interface for retrieving and displaying information stored in the information management system. This includes the visualisation of the simulation results in the form of maps, both in the form of raster (e.g. water depth, bottom elevation, etc.) as well as vector data (e.g. isolines, computational grid, etc.). Additionally, the user is able to interact with the WebGIS application to obtain information such as cross sections, water level time series, etc. at various locations on the map. According to the user's wishes, this information can be provided by the WebGIS application in the form of graphs or tables and at interpolated to various predefined intervals. This implementation, therefore, goes a step further than applications such as WISKI and HydroDesktop, which are primarily focused on time series data for point locations (Ames et al., 2012, KISTERS, 2015), and provides a more general purpose interface for retrieving and analysing hydrological data.

Chapter 6

Summary, evaluation and outlook

6.1 Summary

For this thesis, a general concept and a prototype implementation of a framework for coupling models in hydroinformatic systems has been developed and presented. The framework is inspired by the concepts of tensor objects and OpenMI. It builds upon these concepts and makes use of tensor objects to fully represent information from physical state variables.

The key component of the coupling set-up is the Tensor Exchange Server (TES) which acts as a coupling broker and is responsible for the interactions between the coupled models as well as adapting information to the requirements of the individual coupled models. The TES is an XML-RPC server and communicates with the models using the XML-RPC specification. Since the XML-RPC uses HTTP as the transfer protocol, it makes it possible to couple models in a platform independent way and even over a network connection. Since the adaptation of information is done by the TES, the adaptation methods don't have to implemented by the coupled models. Furthermore, the TES can be adapted to the coupling requirements so that it controls the coupling mechanism to be able to store and later retrieve the relevant information from the other coupled models. This functionality could further be extended to be able to request this information from the coupled models. To prepare the models for coupling using the TES, it is sufficient to link the model against an existing XML-RPC library (or optionally implementing the relevant parts of the XML-RPC specification) and adding methods for interacting with the TES.

The capabilities of the prototype for the coupling models in hydroinformatic systems has been demonstrated with the help of three application examples. These examples demonstrate the ability of the TES for coupling:

- Different types of hydroinformatic models, e.g. hydrodynamic simulation models, metabolism models, database models, GIS models, etc.
- Models with different dimensionalities, i.e. one-, two- and three-dimensional models.
- Models with different time and space discretisations, e.g. implicit or explicit schemes, regular structured grids or irregular grids, etc.
- Models operating on different time and space scales
- Models working with different physical state variables, e.g. pressure field or velocity field, etc.
- Models developed in different programming languages, e.g. C++ or Java
- Models running on different machines and on different platforms such as Linux or Windows

Additionally, a study about the performance overhead when using the coupling framework for the coupling of models has been done in the case of the coupled subsurface and surface flow model.

To summarise, this work builds upon the concepts put forward by the OpenMI concept and advances upon it by:

- representing information as generalised, complete and autonomous tensor objects that are freely transferable among coupled models
- reducing the effort required for coupling models by handling the communication among the models and adaptation of information on their behalf through the coupling broker: the TES

6.2 Evaluation

In chapter 4, a prototype for a coupling framework based on tensor objects where a central entity, the Tensor Exchange Server (TES), acts as an information broker between the coupled models and is responsible for adaptation of information for the

coupled model has been introduced. The applicability of this framework in the coupling of various types of hydroinformatic models has been demonstrated with the help of three application examples. In section 3.1, important goals for the framework for the coupled hydroinformatic models were formulated. Based on the work presented in the subsequent sections, it is now possible to review the feasibility of this framework based on tensor objects for the coupling of models.

- **Completeness**: The representation of information in the form of tensor objects, as introduced in section 3.3 is capable of fully representing information relevant for hydroinformatic models. This information includes dimensions, units, values, coordinate systems, geometry, topology and metadata.

- **Autonomy**: Tensor objects being a complete representation of information, are autonomous and independent of the internal data structures used by individual models. As a result, they can exist independent from the coupled models and can be freely transferred among them.

- **Adaptability**: Tensor objects are capable of adapting themselves to the requirements of the coupled models. Through the use of operators introduced in section 3.4.2, it is also possible to implement additional functions for adapting tensor objects. This functionality has been exploited for the adaptation of information through interpolation, transformation of physical state variables, etc. in the demonstrated application examples.

- **Flexibility and extensibility**: The suitability of using the TES to flexibly couple different types of models such as simulation models, database models, GIS models, etc. has been demonstrated through the various application examples. Also, since the TES acts as the broker for the information, it is possible to modify the coupled model set-up by replacing, adding or removing models according to the coupling requirements. As an example, in the WebGIS application demonstrated in section 5.4, it is possible to replace TELEMAC-2D with HMS, or to extended it by coupling a statistical model for analysing simulation results, if required.

- **Simplicity**: The coupling framework based on tensor objects and the tensor exchange server (TES) is responsible for the communication between the models using web services and for the adaptation of information between different formats provided or required by the coupled models through operations such

as mapping, interpolation, application of the laws of physics, etc. As a result, preparing individual models for coupling is made simpler since these methods don't have to be implemented by the coupled models themselves. Thus, as in the case of DuMu$^{\mathrm{X}}$ and HMS, models can be prepared for coupling by linking them against an XML-RPC library and implementing methods for communicating with the TES for sending and receiving information.

- **Communicability**: The use of XML-RPC for communicating with the TES makes it possible to couple models running on different machines and different operating systems and different hardware types. Therefore, it is possible to couple models in a platform-independent manner.

Coupling of models through tensor objects does have its drawbacks though. These include:

- **Dependencies**: Communication between the TES and the coupled models takes place through the XML-RPC protocol. This means that the coupled models need to be linked against an XML-RPC library, which is an additional dependency for the models. The alternative is for the models to implement the XML-RPC themselves, but this also requires an investment of time and effort. However, other technologies that could possibly be used for communication among the coupled models also have similar limitations.

- **Effort**: Although the coupling framework handles the adaptation of information for the coupled models, the effort required for preparing the models for coupling cannot be fully eliminated. Models still need to be adapted in order to communicate with the TES using XML-RPC for sharing and exchanging information. This is not possible for models for which the source code or a suitable API isn't available.

- **Networking overhead**: XML-RPC communication used by the TES works over the HTTP protocol (UserLand Software, 2003). Models integrated into a software package, on the other hand, are able to directly share the data stored in the computer's memory. Since accessing data over the network is generally much slower than accessing data stored in the computer's memory (Lee Hutchinson, 2012), models coupled through the prototype presented here are expected to be much slower than models integrated into a single application. In our case, the coupling overhead for coupled models running on the same machine was found to be in a range of $-10\,\%$ to $185\,\%$ (see table 5.4).

- **Physics**: despite the best efforts of the modeller, it may not always be possible to properly represent the physical processes involved in the coupling using loosely-coupled models, e.g. in section 5.3.5, one of the reasons for the coupled model being unable to perfectly reproduce laboratory results is though to be the need to better model the physical infiltration processes.

From the author's point of view, the prototype of a model coupling framework based on tensor objects presented in chapter 4 and demonstrated through application examples presented in chapter 5 is a suitable solution for coupling hydroinformatic models where the benefits outweigh the drawbacks. So the coupling approach based on the TES is highly suitable for loosely coupling models in hydroinformatics systems, e.g. web applications for accessing sensor networks showing the location of sensors on the map, displaying the information collected by these sensors as graphs, tables, etc. coupling them with simulation models to provide boundary conditions, etc. However, this approach may not be suitable for models with complex and highly inter-dependent physical interactions such as models simulating interactions between processes like precipitation, evapotranspiration, runoff, sediment transport, etc.

6.3 Outlook

The model coupling framework presented in this work is currently in the prototype stage. Naturally, a number of improvements are possible for the coupling framework. For instance, as part of the current work, only the interfaces describing tensor objects and its components have been defined. It would be useful to standardise the way of representing information in the form of tensor objects with the aid of an XML-based markup standard, called for instance TensorML. Such a standardisation would be useful for models developed in different programming languages and also for archiving the tensor objects and for using them in scenarios such as defining initial and boundary conditions for models with different time and space discretisations.

It would be interesting to extend the capabilities of the TES and enable coupling with other kinds of models, e.g. chemical reaction models using this approach, which might be useful for application to ecological problems.

It would be useful to add support for metadata standards such as ISO 19115 to the model coupling framework in the future. This would be helpful when coupling

models working with spatial data, e.g. simulation models, GIS models, etc.

Support for standards such as WaterML and SensorML would be useful for the coupling framework in order to be able to use hydrological data and the data collected by sensors in coupled models. Similarly, the ability to import data from online services such as PEGELONLINE (WSV, 2015b) and DWD (2015) would be useful for coupled simulations. Such functionality would make it possible to provide coupled models with information, e.g. to set the boundary conditions. A possible scenario would be to extend the WebGIS application presented in section 5.4 as follows:

- Download and save the latest water level, discharge and rainfall data for the stations relevant to the study area from PEGELONLINE and the website of DWD and save it to the information management system.
- Run a hydrodynamic simulation using the latest discharge and water level time series for creating hotstart files to be used for later simulations with newer data and for flood prediction.
- Couple a rainfall-runoff model with the information management system to predict the discharge time series at Ruhrort.
- Run a hydrodynamic simulation using the predicted discharges for doing the flood prediction.
- Convert simulation results to time series, maps, graphs, tables, etc. and save them to the information management system for access through the WebGIS interface.

Additionally, protocols for publishing and automatic discovery of coupling capabilities through the use of services such as WSDL (W3C, 2007b) should be implemented in the coupling framework. Such capabilities might be useful in providing a catalogue of coupled models along with their coupling capabilities. Such a catalogue could in turn be used to provide a modular simulation tool in the form of a web application, where users are able to select different hydroinformatic models, couple them together, run a simulation in a cloud and view or export the results in different formats, all the time using only a browser.

In section 2.3.2, other existing solutions for coupling hydroinformatic models were mentioned. To further facilitate the coupling of models, it would be useful to provide compatibility with these frameworks in the future versions of the coupling framework. Also, similar to the functionality provided by HydroDesktop (Ames

et al., 2012), it would be useful to expand the capabilities of the coupling framework into a full-fledged information management system (similar to the one described in section 5.4) capable of providing access to various different sources of data for use in coupling of models.

For further research, it would also be useful to explore the capabilities of the framework for distributed computing by utilising the networking capabilities of the tensor exchange server. It would also be interesting to investigate whether parallelisation of the Tensor Exchange Server, e.g. by using modern GPUs would speed up the coupling process and by how much. Similarly, the feasibility of using tensor objects for storing, accessing and analysing big data (Wikipedia, 2015a) should be explored.

Code listings

Listing 1: Code for adding coupling support to a one-phase DuMu$^{\mathrm{X}}$ model

```
1  // Regular imports for dumux
2
3  // Imports for adding XML-RPC support
4  #include <xmlrpc-c/base.hpp>
5  #include <xmlrpc-c/client_simple.hpp>
6  #include <xmlrpc-c/girerr.hpp>
7
8  namespace Dumux
9  {
10 template <class TypeTag>
11 class OnePTestProblem;
12
13 template <class typetag>
14 class oneptestproblem
15       : public implicitporousmediaproblem<typetag>
16 {
17     typedef implicitporousmediaproblem<typetag> parenttype;
18     // Other typedefs for dumux
19
20     enum {
21         // grid and world dimension
22         dim = gridview::dimension,
23     };
24
25     typedef std::vector<scalar> vectordouble;
26
27     typedef xmlrpc_c::value value;
28     typedef xmlrpc_c::value_boolean valuebool;
```

```
29      typedef xmlrpc_c::value_string valuestring;
30      typedef xmlrpc_c::value_int valueint;
31      typedef xmlrpc_c::value_double valuedouble;
32      typedef xmlrpc_c::value_array valuearray;
33      typedef xmlrpc_c::paramlist paramlist;
34      typedef xmlrpc_c::rpcptr rpcptr;
35      typedef xmlrpc_c::client_xml client;
36      typedef xmlrpc_c::clientxmltransport_curl transport;
37      typedef xmlrpc_c::carriageparm_curl0 carriageparm;
38      typedef std::vector<value> vectorvalue;
39
40  public:
41      oneptestproblem(timemanager &timemanager,
42                      const gridview &gridview)
43            : parenttype(timemanager, gridview),
44            client_(new transport()),
45            parm_("http://127.0.0.1:8093/xmlrpc")
46      {
47          eps_ = 1.0e-6;
48          patm_ = 101325.0;
49
50          /*
51           * get_runtime_param is a method provided by dumux
52           * for reading input files
53           */
54          string geometrymethod = get_runtime_param(
55                typetag, string, coupled.geometrymethod);
56          string permeabilitymethod = get_runtime_param(
57                typetag, string, coupled.permeabilitymethod);
58          pressuresmethod_ = get_runtime_param(
59                typetag, string, coupled.pressuremethod);
60
61          const int &size = gridview.size(dim);
62          pressures_.resize(size);
63          vectorvalue xids(size);
64          for(int i=0; i<size; ++i)
65          {
66              xids[i] = valueint(i);
67          }
68          ids_ = valuearray(xids);
69
70          // Iterate over grid
71          vectorvalue ids;
```

```
72          vectorvalue coordinates;
73          vectordouble kscalar(size);
74          fvelementgeometry fvelemgeom;
75          elementiterator eit
76                 = this->gridview().template begin<0>();
77          const elementiterator &eitend
78                 = this->gridview().template end<0>();
79          const spatialparams &sp
80                 = this->spatialparams();
81
82          for(; eit != eitend; ++eit)
83          {
84              fvelemgeom.update(gridview, *eit);
85              vectorvalue idsel;
86              vectorvalue coordsel;
87
88              for(int i=0; i<fvelemgeom.numscv; ++i)
89              {
90                  vectorvalue coordsi;
91                  const int &globalidx = this->vertexmapper()
92                       .map(*eit, i, dim);
93                  const globalposition &globalpos
94                       = eit->geometry().corner(i);
95
96                  for(int j=0; j<globalpos.size(); ++j)
97                  {
98                      coordsi.push_back(
99                            valuedouble(globalpos[j]));
100                 }
101                 idsel.push_back(valueint(globalidx));
102                 coordsel.push_back(valuearray(coordsi));
103                 kscalar[globalidx]
104                    = sp.intrinsicpermeability(*eit,
105                                               fvelemgeom,
106                                               i);
107             }
108             ids.push_back(valuearray(idsel));
109             coordinates.push_back(valuearray(coordsel));
110         }
111
112         // Send geometry and topology to coupling broker
113         paramlist geomparams;
114         geomparams.add(valuearray(ids));
```

```
115         geomparams.add(valuearray(coordinates));
116
117         rpcptr geomptr(geometrymethod, geomparams);
118         geomptr->call(&client_, &parm_);
119         assert(geomptr->isfinished());
120
121         // Send parameters to coupling broker
122         vectorvalue permeabilities;
123         for(int i=0; i<kscalar.size(); ++i)
124         {
125             permeabilities.push_back(
126                   valuedouble(kscalar[i]));
127         }
128
129         paramlist kparams;
130         kparams.add(ids_);
131         kparams.add(valuearray(permeabilities));
132
133         rpcptr kptr(permeabilitymethod, kparams);
134         kptr->call(&client_, &parm_);
135         assert(kptr->isfinished());
136     }
137
138     /*
139      * This method is called by dumux at the end of each
140      * time-step. Use this method to send the instantaneous
141      * pressure field to the coupling broker
142      */
143     void postTimeStep()
144     {
145         sendPressures_();
146     }
147
148     /*
149      * Other public methods for setting boundary types,
150      * boundary conditions, initial conditions, etc.
151      */
152
153
154 private:
155     void sendPressures_()
156     {
157         const Scalar &t0 = this->timeManager().time();
```

```
158         const Scalar &dt = this->timeManager().timeStepSize();
159         const Scalar t1 = t0 + dt;
160
161         FVElementGeometry fvElemGeom;
162         VolumeVariables volVars;
163
164         ElementIterator eIt
165                = this->gridView().template begin<0>();
166         const ElementIterator &eItEnd
167                = this->gridView().template end<0>();
168         for(; eIt != eItEnd; ++eIt)
169         {
170             fvElemGeom.update(this->gridView(), *eIt);
171
172             for(int i=0; i<fvElemGeom.numScv; ++i)
173             {
174                 const int &globalIdx = this->vertexMapper()
175                      .map(*eIt, i, dim);
176                 volVars.update(
177                       this->model().curSol()[globalIdx],
178                       *this,
179                       *eIt,
180                       fvElemGeom,
181                       i,
182                       false);
183                 pressures_[globalIdx] = volVars.pressure();
184             }
185         }
186
187         VectorValue pressures;
188         for(int i=0; i<pressures_.size(); ++i)
189         {
190             pressures.push_back(ValueDouble(pressures_[i]));
191         }
192
193
194         ParamList params;
195         params.add(ValueDouble(t1));
196         params.add(ids_);
197         params.add(ValueArray(pressures));
198
199         RpcPtr ptr(pressuresMethod_, params);
200         ptr->call(&client_, &parm_);
```

```
201             assert(ptr->isfinished());
202         }
203
204
205         Scalar eps_;
206         Scalar patm_;
207
208         Client client_;
209         CarriageParm parm_;
210         String pressuresMethod_;
211
212         Value ids_;
213         VectorDouble pressures_;
214     };
215     }
```

Listing 2: Code for adding coupling support to Hz metabolism model

```
1  package hydroinformatics.model;
2
3  import hydroinformatics.coupling.CouplingClient;
4
5  public class RespirationModel {
6
7     private static final double epsilon = 1.0e-6;
8     private static double previous = 0.0;
9
10    private final CouplingClient client = new CouplingClient();
11
12    public static void run(RespirationModelSettings s) {
13
14       // Code for initilisation of the model
15       double[][] faceCentres;
16       double[][] vx, vy;
17
18       // Client handling communication with coupling broker
19       CouplingClient client = new CouplingClient();
20       for(double t=s.tStart; t<s.tEnd; t+=s.dt) {//time loop
21          double[][] vserver;
22          try {
23             vserver = client.getVelocities(t, faceCentres);
24             /*
25              * Loop over velocities from the server and save
26              * them to vx and vy
27              */
28             for(int i=0; i<s.ny; i++) {
29                for(int j=0; j<s.nx; j++) {
30                   vx[i][j] = vserver[i*ny + j][0];
31                   vy[i][j] = vserver[i*ny + j][1];
32                }
33             }
34          } catch (IOException e) {
35             // Exception handling code
36             System.exit(1);
37          }
38          // Rest of operations in the time loop
39
40       }
41    }
```

```
42
43     public static class RespirationModelSettings {
44         private double x0, x1, y0, y1;
45         private int nx, ny;
46         private double tStart, dt, tEnd;
47         private double cInitial, cBoundary;
48         private double globalRespirationRate;
49         private double localRespriationRate;
50
51         // Other methods for setting simulation parameters
52
53     }
54 }
```

Listing 3: XML-RPC client for handling communication between the Hz metabolism model and the TES

```
1  package hydroinformatics.coupling;
2
3  import java.io.IOException;
4  import java.net.MalformedURLException;
5  import java.net.URL;
6
7  // imports handling XML-RPC communication
8  import org.apache.xmlrpc.XmlRpcException;
9  import org.apache.xmlrpc.client.XmlRpcClient;
10 import org.apache.xmlrpc.client.XmlRpcClientConfigImpl;
11 import org.apache.xmlrpc.client.XmlRpcCommonsTransportFactory;
12
13 public class CouplingClient {
14
15    private static final String serverUrl
16          = "http://127.0.0.1:8093/xmlrpc";
17    private static final String methodName
18          = "CouplingHandler.getVelocities";
19
20    private static final int numRetries = 10;
21    private static final long retryInterval = 180000L;
22
23    private final XmlRpcClient client;
24
25    public CouplingClient() {
26
27       final XmlRpcClientConfigImpl config
28             = new XmlRpcClientConfigImpl();
29       try {
30          config.setServerURL(new URL(serverUrl));
31       } catch (MalformedURLException e) {
32          // Exception handling code
33          System.exit(1);
34       }
35       config.setEnabledForExceptions(true);
36       config.setConnectionTimeout(60 * 1000);
37       config.setReplyTimeout(Integer.MAX_VALUE);
38
39       client = new XmlRpcClient();
40       client.setTransportFactory(
```

```
41              new XmlRpcCommonsTransportFactory(client));
42          client.setConfig(config);
43      }
44
45      public double[][] getVelocities(
46              double t,
47              double[][] coords) {
48
49          Exception exception = null;
50          Object[] c = new Object[coords.length];
51          for(int i=0; i<coords.length; i++) {
52              c[i] = coords[i];
53          }
54          final Object[] params = { Double.valueOf(t), c };
55          for (int i=0; i < numRetries; i++) {
56              try {
57                  Object[] o = (Object[]) client.execute(
58                          methodName,
59                          params);
60                  double[][] v = new double[o.length/2][];
61                  for(int j=0; j<o.length; j+=2) {
62                      v[j/2][0] = (Double) o[j];
63                      v[j/2 + 1] = (Double) o[j+1];
64                  }
65                  return v;
66              } catch (XmlRpcException e) {
67                  // Wait for some time before retrying
68                  try {
69                      Thread.sleep(retryInterval);
70                  } catch (final InterruptedException e1) {
71                      // Reset the interrupted status
72                      Thread.currentThread().interrupt();
73                  }
74              }
75          }
76
77          throw new IOException(
78                  "Failed to communicate with server", exception);
79      }
80  }
```

Listing 4: Handler dealing with requests from coupled DuMu^{x} and Hz metabolism models

```
1  package hydroinformatics.coupling;
2
3  import hydroinformatics.xmlrpc.CouplingServer;
4
5  public class CouplingHandler {
6
7     private final TensorSet store = new TensorSet();
8
9     public boolean setCoordinates(
10          Object[] ids,
11          Object[] coordinates) {
12
13       int[][] ii = new int[ids.length][];
14       for(int i=0; i<ids.length; i++) {
15          Object[] o = (Object[]) ids[i];
16          ii[i] = new int[o.length];
17          for(int j=0; j<o.length; j++) {
18             ii[i][j] = (Integer) o[j];
19          }
20       }
21       double[][][] c = new double[coordinates.length][][];
22       for(int i=0; i<coordinates.length; i++) {
23          c[i] = parseDoubleArray(coordinates[i]);
24       }
25       store.setCoordinates(ii, c);
26       return true;
27    }
28
29    public boolean setPermeabilities(
30          Object[] ids,
31          Object[] permeabilities) {
32
33       int[] ii = new int[ids.length];
34       double[] k = new double[permeabilities.length];
35       for(int i=0; i<ids.length; i++) {
36          ii[i] = (Integer) ids[i];
37          k[i] = (Double) permeabilities[i];
38       }
39       store.setPermeabilities(ii, k);
40       return true;
```

```
41     }
42
43     public boolean setPressures(
44           Double time,
45           Object[] ids,
46           Object[] pressures) {
47
48        int[] ii = new int[ids.length];
49        double[] p = new double[pressures.length];
50        for(int i=0; i<ids.length; i++) {
51           ii[i] = (Integer) ids[i];
52           p[i] = (Double) pressures[i];
53        }
54        store.setPressures(time, ii, p);
55        return true;
56     }
57
58     public Object[] getVelocities(
59           Double time,
60           Object[] coordinates) {
61
62        double[][] c = new double[coordinates.length][];
63        for(int i=0; i<coordinates.length; i++) {
64           String[] s = ((String) coordinates[i]).split(":");
65           c[i] = new double[s.length];
66           for(int j=0; j<s.length; j++) {
67              c[i][j] = Double.parseDouble(s[j]);
68           }
69        }
70
71        double[][] v = store.getVelocities(time, c);
72        Object[] result = new Object[v.length * 2];
73        for(int i=0; i<v.length;i++) {
74           result[2 * i] = v[i][0];
75           result[2 * i + 1] = v[i][1];
76        }
77        return result;
78     }
79
80     private double[][] parseDoubleArray(Object o) {
81        Object[] c = (Object[]) o;
82        double[][] result = new double[c.length][];
83        for(int i=0; i<c.length; i++) {
```

```
84            Object[] arr = (Object[]) c[i];
85            result[i] = new double[arr.length];
86            for(int j=0; j<arr.length; j++) {
87               result[i][j] = (Double) arr[j];
88            }
89         }
90         return result;
91      }
92   }
```

Listing 5: Tensor objects for coupling of DuMu^{X} and Hz metabolism model

```
1  package hydroinformatics.information;
2
3  import hydroinformatics.geometry.*;
4  import hydroinformatics.information.*;
5  import hydroinformatics.quantities.*;
6  import hydroinformatics.topology.BasicGrid;
7  import hydroinformatics.util.*;
8
9  import java.util.*;
10 import java.util.concurrent.*;
11
12 public class CouplingTensor implements
13       TensorObject<CalculationNode, ScalarValue, BasicGrid> {
14
15    private final int size;
16    private final CouplingGrid grid;
17    private final ConcurrentNavigableMap<Double, double[]>
18          values;
19
20    /*
21     * Other fields for units, coordinate systems, etc.
22     */
23
24    private final UnaryOperator<Object[], Double>
25          interpolator = BicubicSplineIterpolator.INSTANCE;
26
27    private static final double epsilon = 1.0e-6;
28
29    public CouplingTensor(
30          SIUnit valueUnits,
31          int[] ids,
32          double[] coords) {
33
34       /*
35        * ids from dumux are serial, so we can use ordinary
36        * arrays to map ids to values
37        */
38       int size = 0;
39       for(int id: ids) {
40          size = Math.max(size, id);
41       }
```

```
42          this.size = size;
43
44          grid = new CouplingGrid(ids, coords);
45          values = new ConcurrentSkipListMap<>();
46          this.interpolator = interpolator;
47
48          /*
49           * Initialise other tensor components including units,
50           * coordinate systems, etc.
51           */
52      }
53
54      public void put(int id, double value) {
55          put(0.0, id, value);
56      }
57
58      public void put(Double time, int id, double value) {
59          synchronized (values) {
60              values.putIfAbsent(time, new double[size]);
61              values.get(time)[id] = value;
62              values.notifyAll();
63          }
64      }
65
66      public void get(int id) {
67          return get(0.0, id);
68      }
69
70      public double get(Double time, double[] coords) {
71          synchronized(values) {
72              /*
73               * Check if values are available for time equal to
74               * or greater than the requested time. If not, then
75               * wait for new values from the other model
76               */
77              while(values.ceilingKey(time) == null) {
78                  try{
79                      values.wait();
80                  } catch (IterruptedException e) {
81                      Thread.currentThread().interrupt();
82                  }
83              }
84              Object[] params = { time, values, coords };
```

```
85              return interpolator.operate(params);
86          }
87      }
88
89      /*
90       * Other tensor-related methods
91       */
92
93  }
```

Listing 6: Set of Tensor object for the coupling of DuMu^{X} and Hz metabolism model

```
1  package hydroinformatics.information;
2
3  import hydroinformatics.quantities.*;
4  import hydroinformatics.util.UnaryOperator;
5
6  public class TensorSet {
7
8     private CouplingTensor permeabilities;
9     private CouplingTensor pressures;
10    private CouplingTensor vx;
11    private CouplingTensor vy;
12
13    private final UnaryOperator<Object[], double[][]>
14          interpolator;
15    private final BinaryOperator
16          <CouplingTensor, CouplingTensor, double[][]>
17                operator;
18
19    public TensorSet() {
20
21       interpolator = VelocitiesInterpolator.INSTANCE;
22       operator = DarcysLawOperator.INSTANCE;
23    }
24
25    public void setCoordinates(
26          int[][] ids,
27          double[][][] coords) {
28
29       SIUnit metre = BaseSIUnit.METRE;
30       SIUnit second = BaseSIUnit.SECOND;
31       SIUnit pascal = DerivedSIUnit.PASCAL;
32
33       Dimension dim = metre.pow(2);
34       SIUnit units = BaseSIUnit.getGenericSIUnitFor(dim);
35       permeabilities = new CouplingTensor(
36                                          units,
37                                          ids,
38                                          coords,
39                                          interpolator);
40
41       pressures = new CouplingTensor(
```

```
42                                      pascal,
43                                      ids,
44                                      coords,
45                                      interpolator);
46
47        dim = metre.getDimensions();
48        dim = dim.divide(second.getDimensions());
49        units = BaseSIUnit.getGenericSIUnitFor(dim);
50        vx = new CouplingTensor(
51                                units,
52                                ids,
53                                coords,
54                                interpolator);
55
56        vy = new CouplingTensor(
57                                units,
58                                ids,
59                                coords,
60                                interpolator);
61    }
62
63    public void setPermeabilities(
64          int[] ids,
65          double[] values) {
66
67        // Add values to the tensor
68        for(int i=0; i<ids.length; i++) {
69           permeabilities.put(ids[i], value[i]);
70        }
71    }
72
73    public void setPressures(
74          Double time,
75          int[] ids,
76          double[] values) {
77
78        // Add values to the tensor
79        for(int i=0; i<ids.length; i++) {
80           pressures.put(time, ids[i], values[i]);
81        }
82
83        /*
84         * Delegate the calculation of velocities
```

```
85          * to the relevant operator
86          */
87         double[][] vels
88               = operator.operate(
89                                   permeabilities,
90                                   pressures);
91
92         // Add values to the tensors
93         for(int i=0; i<ids.length; i++) {
94            vx.put(time, ids[i], vels[i][0]);
95            vx.put(time, ids[i], vels[i][1]);
96         }
97      }
98
99      public Double[] getVelocities(
100            Double t,
101            double[][] coords) {
102
103        /*
104         * Delegate the interpolation of velocities
105         * to the relevant operator
106         */
107        Object[] params = { t,
108                            vx,
109                            vy,
110                            coords };
111        return interpolator.operate(params);
112     }
113
114  }
```

Listing 7: Code for adding coupling support to the Richards model in DuMu^{X}

```
1  // include statements for XML-RPC support
2  #include <xmlrpc-c/base.hpp>
3  #include <xmlrpc-c/client_simple.hpp>
4  #include <xmlrpc-c/girerr.hpp>
5
6  namespace Dumux
7  {
8  template <class TypeTag>
9  class RichardsLensProblem : public RichardsBoxProblem<TypeTag>
10 {
11    typedef RichardsBoxProblem<TypeTag> ParentType;
12    // other typedefs for use later
13
14    // Some indices
15    enum {
16       // Index for the continuity equation
17       contiEqIdx = Indices::contiEqIdx,
18
19       // Index for the wetting phase
20       wPhaseIdx = Indices::wPhaseIdx,
21
22       // Grid dimension
23       dim = GridView::dimension
24    }
25
26    typedef std::vector<double> VectorDouble;
27
28    // typedefs for XML-RPC
29    typedef xmlrpc_c::value_array ValueArray;
30    typedef xmlrpc_c::value_double ValueDouble;
31    typedef xmlrpc_c::paramList ParamList;
32    typedef xmlrpc_c::rpcPtr RpcPtr;
33    typedef xmlrpc_c::client Client;
34    typedef xmlrpc_c::clientXmlTransport_curl Transport;
35    typedef xmlrpc_c::carriageParm_curl0 CarriageParm;
36
37    // other code and typedefs for dumux
38
39 public:
40    RichardsLensProblem(TimeManager &timeManager,
41                        const GridView &gridView)
```

```
42          : ParentType(timeManager, gridView),
43                rpcMethod_("exchangeSubsurfaceFlowValues"),
44                client_(new Transport()),
45                parm_("http://localhost:8093/xmlrpc")
46    {
47       int numCoupled;
48       /*
49        * Iterate over grid and calcualte number of
50        * coupled nodes. See lines 166 - 187 for an
51        * example of an iterator in dumux
52        */
53
54        /*
55         * Initialise the water depths to zero for use
56         * in first time step
57         */
58        waterDepths_.resize(numCoupled, 0.0);
59    }
60
61    /*
62     * This method get's called at the end of every time-step
63     */
64    void postTimeStep()
65    {
66       computeFluxes_();
67       doRpc_();
68    }
69
70    /*
71     * Source terms for the primary variables for each element
72     * are queried by the model through this method
73     */
74    void source(PrimaryVariables        &values,
75                const Element           &element,
76                const FvElementGeometry &fvElemGeom,
77                int                     scvIdx)
78    {
79        values = 0.0;
80
81        /*
82         * Check if node lies on the coupled interface.
83         * Return if this is not the case
84         */
```

```
85
86          // Current solution
87          const VertexPointer &vp =
88                elemet.template subEntity<dim>(scvIdx);
89          const int &globalIdx = this->vertexMapper().map(*vp);
90          VolumeVariables volVars;
91          volVars.update(this->model().curSol()[globalIdx],
92                         *this,
93                         element,
94                         fvElemGeom,
95                         scvIdx,
96                         false)
97
98            const double temp = 273.15 + 25;
99            const double &g = this->gravity()[dim - 1];
100           const double &dt = this->timeManager().timeStepSize();
101           const double &density = volVars.density(wPhaseIdx);
102           const Scalar &pw = volVars.pressure(wPhaseIdx)
103           const double waterDepth = waterDepths_[globalIdx];
104           const double &scvVolume =
105                 fvElemGeom.subContVol[scvIdx].volume;
106           double scvHeight;
107           /*
108            * Not shown here is the iteration over sub control
109            * volumes of the element to calculate the height of
110            * the sub control volume: scvHeight
111            */
112           const double scvBreadth = scvVolume / scvHeight;
113
114           const double &head = (pw - patm_) / (density * g);
115           const double &viscosity =
116                 Dumux::H2O<Scalar>::liquidViscosity(temp, pw);
117           const double &permeability =
118                 this->spatialParams()
119                       .intrinsicPermeability(element,
120                                              fvElemGeom,
121                                              scvIdx);
122           const double conductivity =
123                 permeability * g / viscosity;
124
125           const double possible
126                 = -conductivity * head / scvHeight;
127           const double available = waterDepth / dt;
```

```
128
129             double infiltration = std::min(possible, available)
130                                   * scvBreadth
131                                   * 1 // thickness of domain
132                                   * density
133                                   / scvVolume;
134
135             values[contiEqIdx] = infiltration;
136         }
137
138
139         /*
140          * Other public methods including methods for setting
141          * boundary types, boundary conditions, initial
142          * conditions, etc.
143          */
144
145     private:
146         /*
147          * At the end of each time-step, compute the fluxes and
148          * save them to the fluxes_ field to be sent to the server
149          * in the next step
150          */
151         void computeFluxes_()
152         {
153             for(int i=0; i<fluxes_.size(); ++i)
154             {
155                fluxes_[i] = 0.0;
156             }
157            /*
158             * Iterate over the grid elements and save the
159             * fluxes for elements lying on the coupled
160             * interface to the fluxes_ field
161             */
162
163             FVElementGeometry fvElemGeom;
164             ElementVolumeVariables volVars;
165
166             ElementIterator eIt =
167                   this->gridView().template begin<0>();
168             const ElementIterator &eItEnd =
169                   this->gridView().template end<0>();
170             for(; eIt != eItEnd; ++eIt)
```

```
171         {
172             fvElemGeom.update(this->gridView(), *eIt);
173
174             for(int i=0; i<fvElemGeom.numScv; ++i)
175             {
176                 const int &globalIdx =
177                       this->vertexMapper().map(*eIt, i, dim);
178                 volVars.update(*this,
179                                *eIt,
180                                fvElemGeom,
181                                false);
182                 this->model().localResidual()
183                       .evalFluxes(*eIt, volVars);
184                 fluxes_[globalIdx] += this->model().localResidual()
185                       .residual(i)[contiEqIdx];
186             }
187         }
188     }
189
190     /*
191      * Send the flux values to the server. The response from
192      * the server contains the interpolated water-depths. Save
193      * these to the waterDepths_ field for use in calculating
194      * the source terms during the next time-step
195      */
196     void doRpc_()
197     {
198         // Prepare values for RPC
199         ParamList params;
200         const double &t0 = this->timeManager().time();
201         const double &dt =
202               this->timeManager().timeStepSize();
203         const double &t1 = t0 + dt;
204
205         VectorValue coords;
206         /*
207          * Iterate over grid elements and add
208          * coordinates of coupled nodes to the
209          * coords vector
210          */
211          VectorValue fluxes;
212          for(int i=0; i<fluxes_.size(); ++i)
213          {
```

```
214             fluxes.push_back(ValueDouble(fluxes_[i]));
215          }
216
217          params.add(ValueDouble(t1));
218          params.add(ValueArray(coords));
219          params.add(ValueArray(fluxes));
220
221          // Perform RPC
222          RpcPtr ptr(rpcMethod_, params);
223          ptr->call(&client_, &parm_);
224          assert(ptr->isFinished());
225          ValueArray result(ptr->getResult());
226
227          // Save water levels from the server response
228          for(int i=0; i<waterDepths_.size(); ++i)
229          {
230             waterDepths[i] =
231                   ValueDouble(result.vectorValueValue()[i]);
232          }
233       }
234
235       // Other private methods and fields
236
237       VectorDouble fluxes_;
238       VectorDouble waterDepths_;
239
240       const std::string rpcMethod_;
241       Client client_;
242       CarriageParm parm_;
243    };
244    }
```

Listing 8: Code modifications to HMS for adding coupling support to the surface-flow model

```
1  package hydroinformatics.surfaceflow;
2
3  import org.hydroplan.hms.layer.calc.*;
4  import org.hydroplan.hms.layer.surfaceflow.SurfaceFlow;
5  import org.hydroplan.hms.numerics.*;
6  import org.hydroplan.hms.numerics.surfaceflow.*;
7  import org.hydroplan.hms.shape.element.Element;
8  import org.hydroplan.hms.shape.grid.HRegularGridGenerator;
9  import org.hydroplan.hms.shape.mesh.Mesh;
10 import org.hydroplan.hms.system.*;
11 import org.hydroplan.terma.util.Startable;
12
13 public class SurfaceFlowModel implements Startable {
14
15    private CoupledLayer layer;
16    // Other private fields for setting up the model
17
18    public void start(final String... args) {
19       // Parameters for setting up the surface-flow model
20       Domain domain;
21       Mesh mesh;
22       HLayerManager manager;
23       HCalcLayerPreferences layerSetup;
24       DefaultEngineSettings engineSettings;
25       Scheme scheme;
26       /*
27        * Code for initialising parameters and setting up the
28        * surface-flow model
29        */
30
31       DefaultSurfaceFlowSettings surfaceflowSettings
32          = new DefaultSurfaceFlowSettings() {
33
34          @Override
35          public double[] getSource(
36                Element element,
37                TimeStamp t) {
38
39             /*
40              * Delegate the computation of source/sink term
```

```
41                * to the coupled layer
42                */
43               double qSource
44                  = layer.getSource(t.getTime(), element);
45               return new double[] { qSource, 0.0, 0.0 };
46            }
47
48         };
49
50         try {
51            layer = new CoupledLayer(
52                  SurfaceFlow.LAYER_FULL_NAME,
53                  manager,
54                  domain,
55                  mesh,
56                  engine,
57                  SurfaceFlow.META_DATA,
58                  layerSetup);
59         } catch (final Exception e) {
60            // Exception handling code
61            System.exit(1);
62         }
63
64         // Remaining initilisation and setting up code
65
66      }
67
68      public static void main(final String[] args) {
69         Startable appl = new SurfaceFlowModel();
70         appl.start(args);
71      }
72
73   }
```

Listing 9: Overridden surface flow layer in HMS with support for coupling

```
1  package hydroinformatics.surfaceflow;
2
3  import hydroinformatics.benchmark.*;
4  import hydroinformatics.io.xmlrpc.HmsXmlrpcClient;
5
6  import java.io.*;
7  import java.net.MalformedURLException;
8  import java.util.*;
9  import java.util.concurrent.ConcurrentHashMap;
10
11 import org.hydroplan.hms.layer.calc.*;
12 import org.hydroplan.hms.layer.surfaceflow.SurfaceFlow;
13 import org.hydroplan.hms.numerics.*;
14 import org.hydroplan.hms.shape.element.*;
15 import org.hydroplan.hms.shape.grid.*;
16 import org.hydroplan.hms.system.HLayerManager;
17 import org.hydroplan.terma.collection.Iterator;
18 import org.hydroplan.terma.geometry.Point;
19
20 public class CoupledLayer extends HCalcLayer {
21
22    private static final double epsilon = 1.0e-15;
23
24    private final double[] cachedTimestamps = new double[2];
25    private volatile double fluxValuesTimestamp = -1.0;
26
27    private final double cellWidth
28          = CouplingSetupHms.CELL_SIZE;
29
30    private final HmsXmlrpcClient client;
31    private final HLayerManager manager;
32    private final HForwardEulerEngine engine;
33    private final Elements elements;
34
35    private final Double[] coupledCoordinates;
36    private Map<Double, Double> fluxes;
37
38    private final Map<Element, double[]>
39          waterDepthsFromPreviousTimestep;
40
41    public CoupledLayer(
```

```
42          String name,
43          HLayerManager manager,
44          Domain domain,
45          Elements elements,
46          Engine engine,
47          CalcLayerMetaData meta,
48          CalcLayerPreferences prefs)
49          throws MalformedURLException {
50
51       super(name, domain, elements, engine, meta, prefs);
52       client = new HmsXmlrpcClient();
53
54       this.manager = manager;
55       this.elements = elements;
56       waterDepthsFromPreviousTimestep
57             = new HashMap<Element, double[]>();
58
59       /*
60        * Iterate over elements to find the coordinates of
61        * the coupled elements
62        */
63       Set<Double> coupledCoordinateSet
64             = new HashSet<Double>();
65       for (Iterator<? extends Element> it = elements
66             .getIterator(); it.hasNext();) {
67
68          Element element = it.getNext();
69          double x = element.getSpecificPoint().getX();
70          coupledCoordinateSet.add(x);
71
72       }
73       List<Double> coupledCoordinatesList
74             = new ArrayList<Double>(coupledCoordinateSet);
75       coupledCoordinates = coupledCoordinatesList.toArray(
76             new Double[coupledCoordinatesList.size()]);
77       Arrays.sort(coupledCoordinates);
78
79       this.engine = (HForwardEulerEngine) engine;
80    }
81
82    /*
83     * Override the compute method to be able to exchange
84     * information with the coupling broker at every time-
```

```
85       * step
86       */
87      @Override
88      public void compute() {
89
90          double endTimestamp
91                  = manager.getSimulationTime().getTime();
92          double timestepSize = manager.getTimeStep().getTime();
93          double startTimestamp = endTimestamp - timestepSize;
94
95          // Communicate with the coupling broker
96          sendWaterDepths(startTimestamp);
97
98          if (Math.abs(startTimestamp - cachedTimestamps[0])
99                  < epsilon) {
100             startTimestamp = cachedTimestamps[0];
101         }
102
103         int timestepIndex = manager.getCurrentTimeStepCount();
104
105         // run the superclass's compute() method
106         super.compute();
107     }
108
109     private void sendWaterDepths(double time) {
110         List<Double> coords = new ArrayList<Double>();
111         List<Double> values = new ArrayList<Double>();
112         /*
113          * Iterate over elements and send the water depths to
114          * the coupling broker
115          */
116         for (Iterator<? extends Element> iter
117                 = this.getShapes().getShapeIterator();
118                 iter.hasNext();) {
119             Element e = iter.getNext();
120             Point p = e.getSpecificPoint();
121             coords.add(p.getX());
122             double waterDepth = e.getValue(
123                     SurfaceFlow.INDEX_WATER_DEPTH)[0];
124             values.add(waterDepth);
125         }
126         try {
127             client.setWaterDepths(
```

```
128                 time,
129                 coords.toArray(new Double[coords.size()]),
130                 values.toArray(new Double[values.size()]));
131          } catch (Exception e) {
132             // Exception handling code
133             manager.exit();
134          }
135       }
136
137       public synchronized double getSource(
138             double time,
139             Element element) {
140
141          // Get new values from server if required
142          if (fluxValuesTimestamp != time) {
143             getSourcesFromServer(time);
144          }
145
146          double x = element.getSpecificPoint().getX();
147          for (Double coupledCoordinate : coupledCoordinates) {
148             if (Math.abs(x - coupledXCoordinate) < epsilon) {
149                return fluxes.get(coupledCoordinate);
150             }
151          }
152       }
153
154       private void getSourcesFromServer(double time) {
155          if (Math.abs(fluxValuesTimestamp - time) > epsilon) {
156             try {
157                Double[] coords = coupledCoordinates;
158                Double[] values = client.getFluxes(time, coords);
159
160                fluxes = new ConcurrentHashMap<Double, Double>(
161                      values.length);
162                for (int i = 0; i < coords.length; i++) {
163                   fluxes.put(coords[i], values[i]);
164                }
165
166                fluxValuesTimestamp = time;
167             } catch (Exception e) {
168                // Exception handling code
169                manager.exit();
170             }
```

```
171             }
172         }
173
174     }
```

Listing 10: XML-RPC client responsible for comminication between HMS and TES

```
1  package hydroinformatics.io.xmlrpc;
2
3  import hydroinformatics.surfaceflow.CouplingSetupHms;
4
5  import java.net.*;
6
7  // From library handling XML-RPC communication
8  import org.apache.xmlrpc.XmlRpcException;
9  import org.apache.xmlrpc.client.*;
10
11 public class HmsXmlrpcClient {
12
13    private static final String serverUrl
14          = "http://127.0.0.1:8093/xmlrpc";
15    private static final String methodName
16          = "CouplingHandler.exchangeSurfaceFlowValues";
17
18    private static final int numRetries = 10;
19    private static final long retryInterval = 180000L;
20
21    private static final double epsilon = 1.0e-6;
22
23    private double previousTimestamp = -1.0;
24    private Double[] fluxes;
25
26    private final XmlRpcClient client;
27
28    public HmsXmlrpcClient() throws MalformedURLException {
29
30       XmlRpcClientConfigImpl config
31             = new XmlRpcClientConfigImpl();
32       config.setServerURL(new URL(serverUrl));
33       config.setEnabledForExceptions(true);
34       config.setConnectionTimeout(60 * 1000);
35       config.setReplyTimeout(Integer.MAX_VALUE);
36
37       client = new XmlRpcClient();
38       client.setTransportFactory(
39             new XmlRpcCommonsTransportFactory(client));
40       client.setConfig(config);
41    }
```

```
42
43    private Object performRpc(
44          String remoteMethodName,
45          Object... params) {
46
47       for (int i = 0; i < numRetries; i++) {
48          try {
49             return client.execute(remoteMethodName, params);
50          } catch (XmlRpcException e) {
51             exception = e;
52             // Wait for some time before retrying
53             try {
54                Thread.sleep(retryInterval);
55             } catch (InterruptedException e1) {
56                Thread.currentThread().interrupt();
57             }
58          }
59       }
60
61       return null;
62    }
63
64    public void setWaterDepths(
65          Double timestamp,
66          Double[] coordinates,
67          Double[] values) {
68
69       exchangeValues(timestamp, coordinates, values);
70       previousTimestamp = timestamp;
71    }
72
73    public Double[] getFluxes(
74          Double t,
75          Double[] coordinates) {
76
77       if (Math.abs(previousTimestamp - t) < epsilon) {
78          return fluxes;
79       } else {
80          throw new RuntimeException("Sources␣for␣time␣"
81                + t + "␣are␣not␣available.");
82       }
83    }
84
```

```
85     private void exchangeValues(
86           Double t,
87           Double[] coordinates,
88           Double[] waterDepths) {
89
90        if (Math.abs(t - previousTimestamp) > epsilon) {
91           Object[] response = (Object[]) performRpc(
92                 methodName,
93                 t,
94                 coordinates,
95                 waterDepths);
96           updateFluxes(response);
97        }
98     }
99
100    private void updateFluxes(Object[] source) {
101       fluxes = new Double[source.length];
102       System.arraycopy(source, 0, fluxes, 0, source.length);
103    }
104 }
```

Listing 11: Handler dealing with requests from coupled DuMu^{X} and HMS models

```
1  package hydroinformatics.coupling;
2
3  public class CouplingHandler {
4
5     // Tensor objects for coupling
6     private static final TensorSet store =
7           new TensorSet();
8
9     public Double[] exchangeSubsurfaceFlowValues(
10             Double t,
11             Object[] coords,
12             Object[] vals) {
13
14       double[] xs = convertToDoubles(coords);
15       double[] qs = convertToDoubles(vals);
16
17       Double[] hs = store
18             .exchangeSubsurfaceFlowValues(t, xs, qs);
19       return hs;
20    }
21
22    public Object[] exchangeSurfaceFlowValues(
23             Double t,
24             Object[] coords,
25             Object[] vals) {
26       double[] xs = convertToDoubles(coords);
27       double[] qs = convertToDoubles(vals);
28
29       Double[] qs = store
30             .exchangeSurfaceFlowValues(t, xs, hs);
31       return qs;
32    }
33
34    public double[] convertToDoubles(Object[] o) {
35       double[] result = new double[o.length];
36       for(int i=0; i<o.length; i++) {
37          result[i] = (Double) o[i];
38       }
39       return result;
40    }
41 }
```

Listing 12: Tensor objects for coupling of DuMu^{X} and HMS

```
1  package hydroinformatics.information;
2
3  import hydroinformatics.geometry.*;
4  import hydroinformatics.information.*;
5  import hydroinformatics.quantities.*;
6  import hydroinformatics.topology.BasicGrid;
7  import hydroinformatics.util.*;
8
9  import java.util.*;
10 import java.util.concurrent.*;
11
12 public class CouplingTensor implements
13       TensorObject<CalculationNode, ScalarValue, BasicGrid>,
14       BinaryOperator<Double, double[], Double[]> {
15
16    private final ConcurrentNavigableMap<Double, double[][]>
17          values;
18
19    /*
20     * Other fields for units, coordinate systems, grid.
21     */
22
23    private final UnaryOperator<Object[], Double>
24          interpolator;
25
26    private static final double epsilon = 1.0e-6;
27
28    public CouplingTensor(SIUnit valueUnits,
29          UnaryOperator<Object[], Double[]> interpolator) {
30
31       values = new ConcurrentSkipListMap<>();
32       this.interpolator = interpolator;
33
34       /*
35        * Initialise other tensor components including units,
36        * coordinate systems and grid
37        */
38    }
39
40    public void put(Double time, double[][] values) {
41       synchronized (this.values) {
```

```
42            this.values.putIfAbsent(time, values);
43            this.values.notifyAll();
44         }
45      }
46
47      public Double[] operate(Double arg0, double[] arg1) {
48         return interpolator.operate(
49               new Object[] {arg0, values, arg1});
50      }
51
52      /*
53       * Other tensor-related methods
54       */
55
56   }
```

Listing 13: Set of Tensor objects for the coupling of DuMu^X and HMS

```
1  package hydroinformatics.information;
2
3  import hydroinformatics.quantities.*;
4  import hydroinformatics.util.UnaryOperator;
5
6  public class TensorSet {
7
8     private final CouplingTensor waterDepths;
9     private final CouplingTensor fluxVolumes;
10
11    public TensorSet() {
12
13       UnaryOperator<Object[], Double[]> depthInterpolator
14             = TensorWaterDepthInterpolator.INSTANCE;
15       UnaryOperator<Object[], Double[]> fluxInterpolator
16             = TensorFluxInterpolator.INSTANCE;
17
18       BaseSIUnit metre = BaseSIUnit.METRE;
19       Dimension dim = metre.getDimensions().pow(3);
20       dim = dim.divide(BaseSIUnit.SECOND.getDimensions());
21       SIUnit fluxUnits = BaseSIUnit.getGenericSIUnitFor(dim);
22
23       waterDepths = new CouplingTensor(
24             metre, depthInterpolator);
25       fluxVolumes = new CouplingTensor(
26             fluxUnits, fluxInterpolator);
27    }
28
29    public Double[] exchangeSubsurfaceFlowValues(
30          double t,
31          double[] xs,
32          double[] qs) {
33
34       Double key = t;
35       fluxVolumes.put(key, new double[][] {xs, qs});
36       return waterDepths.operate(key, xs);
37    }
38
39    public Double[] exchangeSurfaceFlowValues(
40          double t,
41          double[] xs,
```

```
42              double[] hs) {
43          Double key = t;
44          waterDepths.put(t, new double[][] {xs, hs});
45          return fluxVolumes.operate(key, xs);
46      }
47  }
```

Listing 14: Example of an interpolator for interpolating values in a tensor object in space and time

```
1  package hydroinformatics.information;
2
3  import hydroinformatics.coupling.*;
4  import hydroinformatics.util.UnaryOperator;
5
6  import java.util.Map;
7  import java.util.Map.Entry;
8
9  public enum TensorWaterDepthInterpolator
10       implements UnaryOperator<Object[], Double[]> {
11
12    // Single instance of this interpolator
13    INSTANCE;
14
15    private final Interpolator timeInterpolator
16          = LinearInterpolator.INSTANCE;
17    private final SpaceInterpolator spaceInterpolator
18          = SpaceLinearInterpolator.INSTANCE;
19
20    private static final double epsilon = 1.0e-6;
21
22    public Double[] operate(Object[] arg0) {
23
24       Double key = (Double) arg0[0];
25       @SuppressWarnings("unchecked")
26       Map<Double, double[][]> waterDepths
27             = (Map<Double, double[][]>) arg0[1];
28       double[] xs = (double[]) arg0[2];
29
30       double t = key;
31       double[] h;
32       Entry<Double, double[][]> floor;
33
34       synchronized (waterDepths) {
35          /*
36           * If no values greater than or equal to time t are
37           * avaialable, then wait until the server receives
38           * newer values from the other model
39           */
40          while(waterDepths.ceilingKey(key) == null) {
```

```
41              try {
42                 waterDepths.wait();
43              } catch (InterruptedException e) {
44                 Thread.currentThread().interrupt();
45              }
46           }
47           floor = waterDepths.floorEntry(key);
48           if (Math.abs(t - floor.getKey()) < epsilon) {
49              // Map contains entry. No interpolation needed
50              h = floor.getValue()[1];
51           } else {
52              // Do interpolation in time
53              Entry<Double, double[][]> ceil
54                    = waterDepths.ceilingEntry(key);
55              double t0 = floor.getKey();
56              double t1 = ceil.getKey();
57              double[] h0 = floor.getValue()[1];
58              double[] h1 = ceil.getValue()[1];
59
60              h = new double[h0.length];
61              for(int i=0; i<h0.length; i++) {
62                 h[i] = timeInterpolator
63                       .interpolate(t, t0, h0[i], t1, h1[i]);
64              }
65           }
66        }
67        // Do interpolation in space
68        h = spaceInterpolator
69              .interpolate(xs, floor.getValue()[0], h);
70
71        Double[] result = new Double[h.length];
72        for(int i=0; i<h.length; i++) {
73           result[i] = h[i];
74        }
75        return result;
76     }
77  }
```

Listing 15: JavaScript based WebGIS application

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="Content-Type"
      content="text/thml; charset=utf-8" />
      <title>Rhine WebGIS application</title>

      <!-- stylesheets provided by the libraries used -->
      <link rel="stylesheet" type="text/css"
      href="js/ext/resources/css/ext-all.css">
      </link>
      <link rel="stylesheet" type="text/css"
      href="js/GeoExt/resources/css/geoext-all-debug.css">
      </link>
      <link rel="stylesheet" type="text/css"
      href="js/jqplot/jquery.jqplot.min.css"></link>
      <!-- supplementary (custom) stylesheet -->
      <link rel="stylesheet" type="text/css"
      href="rhine-style.css"></link>

      <!-- Mapping libraries for javascript -->
      <script src="js/OpenLayers/OpenLayers.js"
      type="text/javascript"></script>
      <script src="js/proj4js/lib/proj4js-compressed.js"
      type="text/javascript"></script>

      <!-- GeoExt and Ext for building the GUI -->
      <script src="js/ext/adapter/ext/ext-base.js"
      type="text/javascript"></script>
      <script src="js/ext/ext-all.js"
      type="text/javascript"></script>
      <script src="js/GeoExt/script/GeoExt.js"
      type="text/javascript"></script>

      <!-- XML-RPC support -->
      <script src="js/mimic/mimic.js"></script>

      <!-- jqplot for diplaying graphs -->
      <script src="js/jqplot/jquery.min.js"></script>
      <script src="js/jqplot/jquery.jqplot.min.js"></script>
      <!-- Other jqplot scripts -->
```

```
42
43        <!-- javascript based WebGIS application -->
44        <script type="text/javascript">
45  var epsg_31466 =  new OpenLayers.Projection("EPSG:31466");
46
47  var mapPanel, dataWrapperPanel, leftPanel, graphPanel,
48      timeRadioGroup, dsWaterDepth, dsBathy, tree, legend,
49      statusBar, tblWaterDepth, tblBathy, toolBar, scale,
50      popup, tstart, tnow;
51
52  Ext.onReady(function() {
53     tstart = new Date(1995, 0, 1, 0, 0, 0);
54     tnow = new Date(1995, 0, 1, 0, 0, 0);
55
56     // Some utility functions
57     var formatNumber = function(n) {
58        if (n < 10) {
59           return '0' + n.toString();
60        } else {
61           return n.toString();
62        }
63     }
64     var getTimeString = function(t) {
65        return String.format(
66              '{0}-{1}-{2} {3}:00:00',
67              t.getFullYear(),
68              formatNumber(t.getMonth() + 1),
69              formatNumber(t.getDate()),
70              formatNumber(t.getHours())
71        );
72     };
73     var getLayerName = function(t) {
74        return String.format(
75              'rhine:waterdepth_{0}{1}{2}T{3}0000',
76              t.getFullYear(),
77              formatNumber(t.getMonth() + 1),
78              formatNumber(t.getDate()),
79              formatNumber(t.getHours())
80        );
81     };
82
83     // Layer showing water depths at selected time
84     var wms = "http://lu.umwelt.tu-cottbus.de:8080"
```

```
85          + "/geoserver/rhine/wms";
86      var hLayer = new OpenLayers.Layer.WMS(
87          "Water␣Depths␣1995-01-01␣00:00",
88          wms, {
89              format: "image/png",
90              layers: "rhine:waterdepth_19950101T000000",
91              transparent: true
92          }, {
93              buffer: 0,
94              displayOutsideMaxExtent: true,
95              isBaseLayer: false
96          }
97      );
98
99      var fxnDoRpc = function(method, params) {
100         if (!popup) {
101             popup = new GeoExt.Popup({
102                 collapsible: true,
103                 html: '<p>Loading data.<br />' +
104                     'This might take some time.</p>',
105                 listeners: {
106                     close: function () {
107                         popup = null;
108                     }
109                 },
110                 location: map.getExtent().getCenterLonLat(),
111                 map: map,
112                 title: 'Loading...',
113                 width: 400
114             });
115         }
116         popup.show();
117
118         // Hide GUI elements
119         timeRadioGroup.hide();
120         tblWaterDepth.hide();
121         tblBathy.hide();
122
123         // Display 'loading' message while fetching data
124         var tag = document.getElementById('graph');
125         if (tag) {
126             tag.innerHTML = '<p>Loading data</p>';
127         }
```

```
128
129         // Prepare and perform XML-RPC
130         var path = '/xmlrpc';
131         var request = new XmlRpcRequest(path, method);
132         for(var i=0; i<params.length; i++) {
133            request.addParam(params[i]);
134         }
135         var response = request.send();
136
137         // Get rid of 'loading' message
138         popup.close();
139         popup = null;
140         if (tag) {
141            tag.innerHTML = '';
142         }
143         // Expand panel to display graph and table
144         if (dataWrapperPanel.collapsed) {
145            dataWrapperPanel.expand();
146         }
147         if (graphPanel.collapsed) {
148            graphPanel.expand(true);
149         }
150
151         // Parse response from server and return the result
152         return response.parseXML();
153      }
154
155      dsWaterDepth = new Ext.data.ArrayStore({
156         fields: [
157            { name: 'time'},
158            { name: 'depth', type: 'float' }
159         ]
160      });
161      dsBathy = new Ext.data.ArrayStore({
162         fields: [
163            { name: 'x', type: 'float' },
164            { name: 'z', type: 'float' }
165         ]
166      });
167
168   //#############################################################
169      // Map to be displayed in the main panel
170      var map = new OpenLayers.Map({
```

```
171         allOverlays: false,
172         layers: [
173            // Base map from OpenStreetMap
174            new OpenLayers.Layer.OSM("OpenStreetMap"),
175            new OpenLayers.Layer.WMS("Bathymetry",
176                  wms,{
177                     format: "image/png",
178                     layers: "rhine:bathymetry",
179                     transparent: true
180                  }, {
181                     buffer: 0,
182                     displayOutsideMaxExtent: true,
183                     isBaseLayer: false
184                  }
185            ),
186            hLayer,
187            // Other layers to display in the map
188         ],
189         // handler for the 'zoom to max extents' functionality
190         zoomToMaxExtent: function (e) {
191            map.zoomToExtent(new OpenLayers.Bounds(
192                     [731560, 6701270, 749260, 6737310])
193                     .transform(
194                        new OpenLayers.Projection("EPSG:900913"),
195                        map.getProjectionObject()));
196         }
197      }); // End map
198
199      // Centre the map to the modelled area
200      map.setCenter(
201         new OpenLayers.LonLat(6.6375, 51.549).transform(
202            new OpenLayers.Projection("EPSG:4326"),
203            map.getProjectionObject()
204         )
205      );
206
207   //#############################################################
208      // Controls for the map
209
210      /*
211       * When a user clicks on the map, this custom control:
212       * - finds out the coordinates of the click
213       * - communicates with the XML-RPC server to fetch the
```

```
214      *   water-depth time-series at the location of the click
215      * - displays this information in the form of a graph
216      *   and table
217      */
218     OpenLayers.Control.Click = OpenLayers.Class(
219           OpenLayers.Control, {
220              defaultHandlerOptions: {
221                 'single': true,
222                 'double': false,
223                 'pixelTolerance': 0,
224                 'stopSingle': false,
225                 'stopDouble': false
226              },
227
228              initialize: function(options) {
229                 this.handlerOptions = OpenLayers.Util.extend(
230                       {},
231                       this.defaultHandlerOptions
232                 );
233                 OpenLayers.Control.prototype.initialize.apply(
234                       this, arguments
235                 );
236                 this.handler = new OpenLayers.Handler.Click(
237                       this, {
238                          'click': this.trigger
239                       }, this.handlerOptions
240                 );
241              },
242              trigger: function(e) {
243                 var proj = map.getProjectionObject();
244                 var loc = map.getLonLatFromViewPortPx(e.xy);
245                 var lon = loc.lon.toFixed(2);
246                 var lat = loc.lat.toFixed(2);
247                 loc = loc.transform(proj, epsg_31466);
248                 var freq = 3600 * timeRadioGroup.getValue()
249                       .inputValue;
250
251                 var method = 'WgisHandler.getPointTimeSeries';
252                 var params = [loc.lon, loc.lat, freq]
253
254                 var data = fxnDoRpc(method, params);
255
256                 for(var i=0; i<data.length; i++) {
```

```
257                     var t = new Date(
258                           tstart.getTime() + 1000 * data[i][0]);
259                     data[i][0] = getTimeString(t);
260                 }
261
262                 // Load table data
263                 dsWaterDepth.loadData(data);
264                 // Display the relevant GUI elements
265                 timeRadioGroup.show();
266                 tblWaterDepth.show();
267
268                 // Display the graph
269                 $.jqplot(
270                       'graph',
271                       [data],
272                       {
273                          title:
274                             'Water depth for coordinates ('
275                                   + lon + ', ' + lat + ')',
276                          axes: {
277                             xaxis: {
278                                label: 'Time',
279                                pad: 0,
280                                renderer:
281                                   $.jqplot.DateAxisRenderer,
282                                tickOptions: {
283                                   formatString: '%Y-%m-%d'
284                                }
285                             },
286                             yaxis: {
287                                label: 'Water depth [m]'
288                             }
289                          },
290                          axesDefaults: {
291                             labelRenderer: $.jqplot
292                                   .CanvasAxisLabelRenderer
293                          },
294                          series: [{
295                             markerOptions: { show: false }}]
296                       });
297             }
298
299     });
```

```
300
301     /*
302      * This controler overrides the functionality of the
303      * measure control provided by OpenLayers to display the
304      * cross-section of the bathymetry data as a graph and a
305      * table
306      */
307     var ctrlLength = new OpenLayers.Control.Measure(
308           OpenLayers.Handler.Path, {
309              eventListeners: {
310                 measure: function (e) {
311                    var proj = map.getProjectionObject();
312                    var len = e.measure.toFixed(2);
313                    var units = e.units;
314                    var vertices = e.geometry.getVertices();
315                    var coords = [];
316                    for(var i=0; i<vertices.length; i++) {
317                       var vert = vertices[i].transform(
318                             proj, epsg_31466);
319                       coords.push(vert.x);
320                       coords.push(vert.y);
321                    }
322
323                    var method
324                       = 'WgisHandler.getBathymetryProfile';
325                    var params = [coords];
326                    var data = fxnDoRpc(method, params);
327
328                    // Load table data
329                    dsBathy.loadData(data);
330                    // Display the relevant GUI elements
331                    tblBathy.show();
332
333                    // Display the graph
334                    $.jqplot('graph', [data], {
335                       title: 'Bathymetry profile',
336                       axes: {
337                          xaxis: {
338                             label: 'Distance [m]',
339                             pad: 0,
340                          },
341                          yaxis: { label: 'Elevation [m]' }
342                       },
```

```
343                     axesDefaults: {
344                        labelRenderer: $.jqplot
345                              .CanvasAxisLabelRenderer
346                     },
347                     series: [{
348                        markerOptions: { show: false }}]
349                  });
350               }
351            },
352            persist: true
353    });
354    // Additional controls
355    var ctrlNav = new OpenLayers.Control.NavigationHistory();
356    map.addControl(new OpenLayers.Control.ScaleLine(
357             {maxWidth: 120}));
358    map.addControl(ctrlLength);
359    map.addControl(ctrlNav);
360
361 //############################################################
362    // UI Elements
363
364    // Slider updating the water-levels layer
365    var slider = new Ext.Slider({
366       maxValue: 117,
367       minValue: 0,
368       width: 200
369    });
370    // Event listener for updating the date-time string
371    slider.on('change',
372          function(slider, newValue, thumb) {
373             tnow.setTime(tstart.getTime()
374                   + 43200000 * thumb.value);
375             document.getElementById("tnow").innerHTML
376                   = getTimeString(tnow);
377    });
378    // Event listener for updating the water-depths layer
379    slider.on('changecomplete',
380          function(slider, newValue, thumb) {
381             // remove the old layer
382             map.removeLayer(hLayer, false);
383             hLayer.destroy();
384
385             var llabel = "Water␣Depths␣"
```

```
386                  + getTimeString(tnow);
387             var lname = getLayerName(tnow);
388             // Re-initialise layer and add it to map
389             hLayer = new OpenLayers.Layer.WMS(llabel,
390                  wms,
391                  {
392                     format: "image/png",
393                     layers: lname,
394                     transparent: true
395                  }, {
396                     buffer: 0,
397                     displayOutsideMaxExtent: true,
398                     isBaseLayer: false
399                  }
400             );
401             map.addLayer(hLayer);
402
403    });
404    // HTML markup for the status bar
405    var statusBar = [
406       '->',
407       '<p id="statusText"> </p>',
408       '-',
409       '<p id="scaleText"> </p>'
410    ];
411
412    // Remaining code for creating and displaying UI elements
413
414 //#############################################################
415    // Events
416    var setStatus = function(pos) { // pos: OpenLayers.LonLat
417       var lon = pos.lon.toFixed(2);
418       var lat = pos.lat.toFixed(2);
419       var proj = map.getProjectionObject();
420       var units = proj.getUnits();
421
422       var text = "Easting␣=␣" + lon + units;
423       text += ",␣Northing␣=␣" + lat + units;
424       text += "␣(" + proj.toString() + ")";
425
426       var tag = document.getElementById("statusText");
427       if (tag) {
428          tag.innerHTML = text;
```

```
429         }
430      };
431
432      // Update coordinates in the status bar
433      map.events.register("mousemove", map, function (e) {
434         var pos = this.events.getMousePosition(e);
435         var lonlat = map.getLonLatFromPixel(pos);
436         setStatus(lonlat);
437      });
438      // Update the scale in status bar
439      map.events.register("zoomend", map, function (e) {
440         var tag = document.getElementById("scaleText");
441         if (tag) {
442            tag.innerHTML = "Scale␣=␣1:"
443                  + map.getScale().toFixed(1);
444         }
445      });
446
447   //############################################################
448      // Initialise the status bar
449      setStatus(map.getCenter());
450      document.getElementById("scaleText").innerHTML
451         = "Scale␣=␣" + map.getScale().toFixed(3);
452      // Initialise the time string in the toolbar
453      document.getElementById("tnow").innerHTML
454            = getTimeString(tnow);
455      // Clear navigation history
456      ctrlNav.clear();
457   });
458
459         </script>
460      </head>
461      <body>
462         <div id="desc">
463         <!-- HTML code for description to be displayed in the
464            right column -->
465         </div>
466         <div id="graph">
467         <!-- div for displaying graphs generated by jqplot -->
468         </div>
469      </body>
470   </html>
```

Listing 16: Operator for retrieving time-series from PostgreSQL database

```
1  package hydroinformatics.xmlrpc;
2
3  import hydroinformatics.util.*;
4
5  import java.sql.*;
6  import java.util.*;
7
8  public class WaterDepthOperator
9        implements BinaryOperator<Double, Double, Double[][]> {
10
11    // Database credentials
12    private static final String url
13          = "jdbc:postgresql://localhost/rhine_postgis";
14    private static final String user = "rhine_postgis";
15    private static final String pw = "hunter2";
16
17
18    public Double[][] operate(Double xCoord, Double yCoord) {
19       double x = (Double) params[0];
20       double y = (Double) params[1];
21       String pt = "'SRID=31466;POINT(" + x + " " + y
22             + ")'::geometry";
23
24       try (Connection con
25             = DriverManager.getConnection(url, user, pw)) {
26          String q = "SELECT * " +
27                "FROM node_ids AS g " +
28                "WHERE " +
29                "ST_DWithin(ST_SetSRID(g.the_geom, 31466), " +
30                pt + ", 250) " +
31                "ORDER BY g.the_geom <-> " + pt + " " +
32                "LIMIT 1";
33          String qDepth = "SELECT (d.unnest).\"time\", " +
34                "(d.unnest).value " +
35                "FROM (SELECT unnest(s.water_depth) " +
36                "FROM simulation_results AS s " +
37                "WHERE s.node_id = %s) AS d";
38
39          Double[][] result = null;
40          /*
41           * Create and run the query to get the nearest-
```

```
42            * neighhbour of the given point for which
43            * simulation results are available
44            */
45           Statement ps = con.createStatement();
46           ResultSet rs = ps.executeQuery(q);
47
48           if (rs.next()) {
49              /*
50               * Get id of the nearest neighbour and retrieve
51               * and return the time-series for that node
52               */
53              String id = rs.getString(1);
54              String query = String.format(qDepth, id);
55              ResultSet rs1 = ps.executeQuery(query);
56              List<Double[]> vals = new ArrayList<>();
57              while(rs1.next()) {
58                 vals.add(new Double[] {
59                    rs1.getDouble(1),
60                    rs1.getDouble(2)
61                 });
62              }
63
64              result = new Double[vals.size()][];
65              for(int i=0; i<result.length; i++) {
66                 result[i] = vals.get(i);
67              }
68
69              return result;
70           }
71
72           rs.close();
73           ps.close();
74           return result;
75        } catch (SQLException e) {
76           // Exception handling code
77        }
78     }
79
80  }
```

Listing 17: Operator for retrieving bathymetry profile from PostgreSQL database

```
1  package hydroinformatics.xmlrpc;
2
3  import hydroinformatics.util.*;
4
5  import java.sql.*;
6  import java.util.*;
7
8  public class WaterDepthOperator
9        implements UnaryOperator<Object[], Double[][]> {
10
11    // Database credentials
12    private static final String url
13          = "jdbc:postgresql://localhost/rhine_postgis";
14    private static final String user = "rhine_postgis";
15    private static final String pw = "hunter2";
16
17
18    public Double[][] operate(Object[] coords) {
19       // construct the points and lines
20       double[][] pts = new double[coords.length/2][2];
21       for(int i=0; i<coords.length; i+=2) {
22          pts[i/2][0] = (double) coords[i];
23          pts[i/2][1] = (double) coords[i+1];
24       }
25       double[][] lines = new double[pts.length-1][4];
26       for(int i=0; i<pts.length-1; i++) {
27          lines[i][0] = pts[i][0];
28          lines[i][1] = pts[i][1];
29          lines[i][2] = pts[i+1][0];
30          lines[i][3] = pts[i+1][1];
31       }
32
33       Double[][] result = new Double[0][];
34       if (pts.length < 1) {
35          return result;
36       }
37
38       try (Connection con
39                = DriverManager.getConnection(url, user, pw)) {
40          /*
41           * Query to find out points that lie within the
```

```
42            * computational domain
43            */
44           String qWithin = "␣ST_Within(" +
45                 "'SRID=31466;POINT(%f␣%f)'" +
46                 "::geometry,␣g.st_polygon)";
47           /*
48            * Query to create line from the coordinates of its
49            * end nodes
50            */
51           String qLine = "␣ST_LineFromText("+
52                 "'LINESTRING(%f␣%f,␣%f␣%f)',␣31466)";
53           /*
54            * Query to check which of the given points lie in
55            * the computational domain
56            */
57           StringBuilder q = new StringBuilder();
58           q.append("SELECT").append(
59                 String.format(qWithin, pts[0][0], pts[0][1]));
60           for(int i=1; i<pts.length; i++) {
61              q.append(',').append(String
62                    .format(qWithin, pts[i][0], pts[i][1]));
63           }
64           q.append("␣FROM␣(");
65           q.append("SELECT␣ST_Polygon(gg.the_geom,␣31466)");
66           q.append("␣FROM␣computational_domain␣AS␣gg");
67           q.append(")␣AS␣g");
68
69           Statement st = con.createStatement();
70           ResultSet rs = st.executeQuery(q.toString());
71
72           boolean[] within = new boolean[pts.length];
73           if (rs.next()) {
74              for(int i=0; i < within.length; i++) {
75                 within[i] = rs.getBoolean(i+1);
76              }
77           }
78
79           rs.close();
80           st.close();
81
82           double x = 0.0;
83           List<Double[]> list = new ArrayList<>();
84           /*
```

```
85            * Query to find out the elevetion of the end-node
86            * of a line
87            */
88           String qEndNode = "SELECT␣n.elevation␣" +
89                 "FROM␣computational_node_elevations␣AS␣n,␣" +
90                 "(SELECT␣pt␣FROM␣(" +
91                 "SELECT␣ST_Polygon(gg.the_geom,␣31466)␣" +
92                 "FROM␣computational_domain␣AS␣gg)␣AS␣g,␣" +
93                 "ST_PointFromText('POINT(%f␣%f)',␣" +
94                 "31466)␣AS␣pt␣" +
95                 "WHERE␣ST_Within(pt,␣g.st_polygon))␣AS␣p␣" +
96                 "WHERE␣ST_DWithin(" +
97                 "ST_SetSRID(n.the_geom,␣31466),␣p.pt,␣250)␣" +
98                 "ORDER␣BY␣n.the_geom␣<->␣p.pt␣" +
99                 "LIMIT␣1";
100          // Query to compute the distance between two points
101          String qDist = "SELECT␣ST_Distance(a,␣b)␣" +
102                "FROM␣␣ST_PointFromText('%s',␣31466)␣AS␣a,␣" +
103                "ST_PointFromText('%s',␣31466)␣AS␣b";
104          /*
105           * Iterate over the lines to figure out intersection
106           * points of the line and the edges of the
107           * computational grid
108           */
109          for(int i=0; i<lines.length; i++) {
110             q = new StringBuilder();
111             q.append("SELECT␣g.elevation1,␣g.elevation2,␣");
112             q.append("ST_Distance(g.vertex1,␣i),␣");
113             q.append("ST_Distance(g.vertex2,␣i),␣");
114             q.append("ST_AsText(i)");
115
116             q.append("␣FROM␣(");
117             q.append("SELECT␣*␣"
118                   + "FROM␣mesh_edges_elevations␣AS␣gg,␣");
119             q.append(String.format(
120                      qLine,
121                      lines[i][0],
122                      lines[i][1],
123                      lines[i][2],
124                      lines[i][3]));
125             q.append("␣AS␣line");
126             q.append("␣WHERE␣ST_Intersects("
127                   + "line,␣gg.the_geom))␣AS␣g,");
```

```
128             q.append("␣ST_Intersection(line,␣g.the_geom");
129             q.append(")␣AS␣i␣");
130             // Sorting order for the nodes
131             if (lines[i][0] < lines[i][2]) {
132                q.append("ORDER␣BY␣ST_X(i)");
133             } else if (lines[i][0] > lines[i][2]) {
134                q.append("ORDER␣BY␣ST_X(i)␣DESC");
135             } else if (lines[i][1] < lines[i][3]) {
136                q.append("ORDER␣BY␣ST_Y(i)");
137             } else if (lines[i][1] > lines[i][3]) {
138                q.append("ORDER␣BY␣ST_Y(i)␣DESC");
139             }
140
141             // First line is handled differently
142             if (i == 0) {
143                st = con.createStatement();
144                rs = st.executeQuery(String.format(
145                         qEndNode, lines[i][0], lines[i][1]));
146
147                if (rs.next()) {
148                   list.add(
149                         new Double[] { x, rs.getDouble(1) });
150                } else {
151                   list.add(new Double[] { x, 0.0 });
152                }
153
154                rs.close();
155                st.close();
156             }
157
158             /*
159              * Iterate over intersection points to figure
160              * out the elevations at these points through
161              * linear interpolation
162              */
163             // First end node of line
164             List<String> verts = new ArrayList<>();
165             verts.add(String.format(
166                      "POINT(%f␣%f)",
167                      lines[i][0],
168                      lines[i][1]));
169             st = con.createStatement();
170             rs = st.executeQuery(q.toString());
```

```
171             // Intermediate intersection points
172             while(rs.next()) {
173                double y1 = rs.getDouble(1);
174                double y2 = rs.getDouble(2);
175                double x1 = rs.getDouble(3);
176                double x2 = rs.getDouble(4);
177                double e = y1 + ((y2 -y1) * x1 /(x2 + x1));
178
179                String pt1 = verts.get(verts.size() - 1);
180                String pt2 = rs.getString(5);
181                verts.add(pt2);
182
183                Statement st1 = con.createStatement();
184                ResultSet rs1 = st1.executeQuery(
185                      String.format(qDist, pt1, pt2));
186
187                if (rs1.next()) {
188                   x += rs1.getDouble(1);
189                   list.add(new Double[] { x, e });
190                }
191
192                rs1.close();
193                st1.close();
194             }
195             // Second end node of line
196             String pt1 = verts.get(verts.size() - 1);
197             String pt2 = String.format(
198                   "POINT(%f␣%f)", lines[i][2], lines[i][3]);
199             verts.add(pt2);
200
201             rs.close();
202             st.close();
203
204             st = con.createStatement();
205             rs = st.executeQuery(String.format(
206                      qEndNode, lines[i][2], lines[i][3]));
207
208             double e = 0.0;
209             if (rs.next()) {
210                e = rs.getDouble(1);
211             }
212
213             rs.close();
```

```
214             st.close();
215
216
217             st = con.createStatement();
218             rs = st.executeQuery(
219                   String.format(qDist, pt1, pt2));
220
221             if (rs.next()) {
222                x += rs.getDouble(1);
223                list.add(new Double[] { x, e });
224             }
225
226             rs.close();
227             st.close();
228          }
229          result = new Double[list.size()][];
230          for(int i=0; i<list.size(); i++) {
231             result[i] = list.get(i);
232          }
233       } catch (SQLException e) {
234          // Exception handling code
235       }
236
237       return result;
238    }
239
240 }
```

Bibliography

Abbott, M., Bathurst, J., Cunge, J., O'Connell, P., and Rasmussen, J. An introduction to the European Hydrological System – Systeme Hydrologique Europeen, "SHE", 1: History and philosophy of a physically-based, distributed modelling system. *Journal of Hydrology*, 87(1-2):45 – 59, 1986. ISSN 0022-1694. doi: 10.1016/0022-1694(86)90114-9. URL http://www.sciencedirect.com/science/article/B6V6C-487D1XS-5Y/2/4c59ce9b2490772ef8c87f95125cdfae.

Alcrudo, F. and Garcia-Navarro, P. A high-resolution godunov-type scheme in finite volumes for the 2d shallow-water equations. *International Journal for Numerical Methods in Fluids*, 16(6):489–505, 1993. ISSN 1097-0363. doi: 10.1002/fld.1650160604. URL http://dx.doi.org/10.1002/fld.1650160604.

Ames, D. P., Horsburgh, J. S., Cao, Y., Kadlec, J., Whiteaker, T., and Valentine, D. HydroDesktop: Web services-based software for hydrologic data discovery, download, visualization, and analysis. *Environmental Modelling & Software*, 37(0):146 – 156, 2012. doi: 10.1016/j.envsoft.2012.03.013. URL http://www.sciencedirect.com/science/article/pii/S1364815212001053.

Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., and Sorensen, D. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999. ISBN 0-89871-447-8 (paperback).

Apache Commons. Commons Math: The Apache Commons Mathematics Library, Apr 2015. URL http://commons.apache.org/proper/commons-math/. Accessed: 2015-03-16.

Argerich, A., Haggerty, R., Martí, E., Sabater, F., and Zarnetske, J. *Quantification of metabolically active transient storage (MATS) in two reaches with contrasting transient storage and ecosystem respiration*, volume 116. 2011.

Bancilhon, F., Delobel, C., and Kanellakis, P. *Building an Object-Oriented Database System: The Story of O2*. Morgan Kaufmann, 1992. ISBN 9781558601697.

Bastian, P., Blatt, M., Dedner, A., Engwer, C., Klöfkorn, R., Ohlberger, M., and Sander, O. A generic grid interface for parallel and adaptive scientific computing. part i: abstract framework. *Computing*, 82:103–119, 2008. ISSN 0010-485X. doi: 10.1007/s00607-008-0003-x. URL http://dx.doi.org/10.1007/s00607-008-0003-x.

Bezroukov, N. Pipes – powerful and elegant programming paradigm, Dec 2014. URL http://www.softpanorama.org/Scripting/pipes.shtml. Accessed: 2015-03-16.

Boisvert, E. and Brodaric, B. Groundwater markup language (gwml) – enabling groundwater data interoperability in spatial data infrastructures. *Journal of Hydroinformatics*, 14(1):93–107, Jan 2012. doi: 10.2166/hydro.2011.172. URL http://www.iwaponline.com/jh/014/jh0140093.htm.

Boulton, A. J., Findlay, S., Marmonier, P., Stanley, E. H., and Valett, H. M. The functional significance of the hyporheic zone in streams and rivers. *Annual Review of Ecology and Systematics*, 29(1):59–81, 1998. doi: 10.1146/annurev.ecolsys.29.1.59. URL http://dx.doi.org/10.1146/annurev.ecolsys.29.1.59.

Bridgman, P. W. *Dimensional Analysis*. Yale University Press, 1922. ISBN 9780548910290.

Brightmix. Brightmix, 2014. URL https://www.iconfinder.com/icons/43239/earth_monotone_world_icon. Accessed: 2013-03-20.

Buckingham, E. On physically similar systems; illustrations of the use of dimensional equations. *Physical Review*, 4:345–376, Oct 1914. doi: 10.1103/PhysRev.4.345.

Bureau International des Poids et Mesures. *International System of Units (SI)*. Bureau International des Poids et Mesures, 2006. ISBN 9789282222133. URL http://www.bipm.org/utils/common/pdf/si_brochure_8_en.pdf.

Busse, T., Simons, F., Mieth, S., Hinkelmann, R., and Molkenthin, F. HMS: a generalised software design to enhance themodelling of geospatial referenced flow and transportphenomena. In *Proceedings of the 10th International Conference onHydroinformatics. Hamburg, Germany*, 2012.

Butler, H., Schmidt, C., Springmeyer, D., and Livni, J. DHDN / 3-degree Gauss-Kruger zone 2, 2015. URL http://spatialreference.org/ref/epsg/31466/. Accessed: 2015-03-16.

Böhlke, J., Antweiler, R., Harvey, J., Laursen, A., Smith, L., Smith, R., and Voytek, M. Multi-scale measurements and modeling of denitrification in streams with varying flow and nitrate concentration in the upper mississippi river basin, usa. *Biogeochemistry*, 93:117–141, 2009. ISSN 0168-2563. doi: 10.1007/s10533-008-9282-8. URL http://dx.doi.org/10.1007/s10533-008-9282-8.

Carey, G. *Computational Grids: Generations, Adaptation & Solution Strategies*. Series in Computational and Physical Processes in Mechanics and Thermal Sciences. Taylor & Francis Group, 1997. ISBN 9781560326359.

Cea, L., Legout, C., Darboux, F., Esteves, M., and Nord, G. Experimental validation of a 2d overland flow model using high resolution water depth and velocity data. *Journal of Hydrology*, 513(0):142 – 153, 2014. ISSN 0022-1694. doi: http://dx.doi.org/10.1016/j.jhydrol.2014.03.052. URL http://www.sciencedirect.com/science/article/pii/S0022169414002376.

CETMEF. CETMEF, 2015. URL http://prepro.fudaa.fr/. Accessed: 2015-03-16.

Chappell, D. A. and Jewell, T. *Java Web Services*. Java Series. Oreilly & Associates Incorporated, 2002. ISBN 978-0-596-00269-5. URL http://shop.oreilly.com/product/9780596002695.do.

Choi, J., Harvey, J., and Conklin, M. Characterizing multiple timescales of stream and storage zone interaction that affect solute fate and transport in streams. *Water Resources Research*, 36:1511–1518, 2000. doi: 10.1029/2000WR900051. URL http://dx.doi.org/10.1029/2000WR900051.

Chua, L. H. C., Merting, F., and Holz, K.-P. River inundation modelling for risk analysis. In *Proceedings of 2. Forum Katastrophenvorsorge "Extreme Naturereignisse - Folgen, Vorsorge, Werkzeuge", 2001, Leipzig*, pages 286–293, 2001.

Codd, E. F. *The Relational Model for Database Management: Version 2*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990. ISBN 0-201-14192-2.

ComputerWeekly.com. Write once, run anywhere?, May 2002. URL `http://www.computerweekly.com/feature/Write-once-run-anywhere`. Accessed: 2015-03-16.

Creative Commons. Creative Commons - About the licenses, 2015. URL `https://creativecommons.org/licenses/`. Accessed: 2015-03-16.

Crockford, D. RFC 4627 - The application/json Media Type for JavaScript Object Notation (JSON), Jul 2006. URL `http://tools.ietf.org/html/rfc4627`. Accessed: 2015-03-16.

CUAHSI-HIS Project Team. CUAHSI Hydrologic Information System (CUAHSI-HIS), 2015. URL `http://his.cuahsi.org/`. Accessed: 2015-03-16.

David, O., II, J. A., Lloyd, W., Green, T., Rojas, K., Leavesley, G., and Ahuja, L. A software engineering perspective on environmental modeling framework design: The object modeling system. *Environmental Modelling & Software*, 39 (0):201 – 213, 2013. doi: 10.1016/j.envsoft.2012.03.006. URL `http://www.sciencedirect.com/science/article/pii/S1364815212000886`.

DCMI. Dublin Core Metadata Element Set, Version 1.1, Jun 2012a. URL `http://dublincore.org/documents/dces/`. Accessed:2015-03-01.

DCMI. DCMI Metadata Terms, Jun 2012b. URL `http://dublincore.org/documents/dcmi-type-vocabulary/`. Accessed: 2015-03-16.

DCMI. Dublic Core Metadata Initiative (DCMI) Home Page, 2015a. URL `http://dublincore.org/index.shtml.rdf`. Accessed: 2015-03-16.

DCMI. DCMI Metadata Basics, 2015b. URL `http://dublincore.org/metadata-basics/`. Accessed: 2015-03-16.

Debian Project. Debian, 2015. URL `https://www.debian.org/`. Accessed: 2015-03-16.

Delfs, J.-O., Park, C.-H., and Kolditz, O. A sensitivity analysis of hortonian flow. *Advances in Water Resources*, 32(9):1386 – 1395, 2009. ISSN 0309-1708. doi: 10.1016/j.advwatres.2009.06.005. URL http://www.sciencedirect.com/science/article/pii/S0309170809000955.

Dilley, B. jsonrpc4j, 2015. URL https://github.com/briandilley/jsonrpc4j. Accessed: 2015-03-16.

DuMu$^{\text{X}}$. *DuMu^{X} Handbook*, 2014. URL http://www.dune-project.org/doc/grid-howto/grid-howto.pdf.

DUNE. Dune Numerics, 2015. URL http://www.dune-project.org/. Accessed: 2015-03-16.

DWD. Weather and Climate, 2015. URL http://www.dwd.de/bvbw/appmanager/bvbw/dwdwwwDesktop?_nfpb=true. Accessed: 2015-03-16.

Elosegi, A., Díez, J., and Mutz, M. Effects of hydromorphological integrity on biodiversity and functioning of river ecosystems. *Hydrobiologia*, 657:199–215, 2010. ISSN 0018-8158. doi: 10.1007/s10750-009-0083-4. URL http://dx.doi.org/10.1007/s10750-009-0083-4.

Enquist, B., Economo, E., Huxman, T., Allen, A., Ignace, D., and Gillooly, J. Scaling metabolism from organisms to ecosystems. *Nature*, 423:639–642, 2003. URL http://www.nature.com/nature/journal/v423/n6940/full/nature01671.html.

ESRI. ArcGIS for Desktop | Key Features, Jul 2013a. URL http://web.archive.org/web/20131001114758/http://www.esri.com/software/arcgis/arcgis-for-desktop/features. Accessed: 2013-10-27.

ESRI. ArcGIS for Server | Functionality Table, 2013b. URL http://web.archive.org/web/20131001193341/http://www.esri.com/software/arcgis/arcgisserver/features/functionality-table. Accessed: 2013-10-27.

ESRI. Overview | Geographic Information Systems, 2013c. URL http://web.archive.org/web/20131001114155/http://www.esri.com/what-is-gis/overview#overview_panel. Accessed: 2013-03-27.

ESRI. ArcGIS Platform, 2015. URL http://www.esri.com/software/arcgis/. Accessed: 2015-03-16.

EUR-Lex. Directive 2007/2/EC of the European Parliament and of the Council of 14 March 2007 establishing an Infrastructure for Spatial Information in the European Community (INSPIRE), Mar 2007. URL http://eur-lex.europa.eu/legal-content/EN/ALL/?uri=CELEX:32007L0002. Accessed: 2015-03-16.

EUR-Lex. Commission Regulation (EC) No 1205/2008 of 3 December 2008 implementing Directive 2007/2/EC of the European Parliament and of the Council as regards metadata (Text with EEA relevance), Dec 2008. URL http://eur-lex.europa.eu/legal-content/EN/ALL/?uri=CELEX:32008R1205. Accessed: 2015-03-16.

European Union. The EU Water Framework Directive - integrated river basin management for Europe, 2000. URL http://ec.europa.eu/environment/water/water-framework/index_en.html. Accessed: 2015-03-16.

European Union. A European Flood Action programme, 2007a. URL http://ec.europa.eu/environment/water/flood_risk/flood_risk.htm. Accessed: 2015-03-16.

European Union. A new EU Floods Directive, 2007b. URL http://ec.europa.eu/environment/water/flood_risk/index.htm. Accessed: 2015-03-16.

European Union. Infrastructure for Spatial Information in the European Community, 2007c. URL http://inspire.ec.europa.eu/index.cfm/pageid/48. Accessed: 2015-03-16.

European Union. Infrastructure for Spatial Information in the European Community, 2007d. URL http://inspire.ec.europa.eu/. Accessed: 2015-03-16.

Fellows, C. S., Valett, H. M., and Dahm, C. N. Whole-stream metabolism in two montane streams: Contribution of the hyporheic zone. *Limnology and Oceanography*, 46:523–531, 2001. doi: 10.4319/lo.2001.46.3.0523.

Ferziger, J. H. and Perić, M. *Computational Methods for Fluid Dynamics*. Springer, 3rd edition, 2002.

Flemisch, B., Darcis, M., Erbertseder, K., Faigle, B., Lauser, A., Mosthaf, K., Müthing, S., Nuske, P., Tatomir, A., Wolff, M., and Helmig, R. DuMu^x: DUNE for multi-{phase, component, scale, physics, ...} flow and transport in porous media. *Advances in Water Resources*, 34(9):1102 – 1112, 2011. doi: 10.1016/j.advwatres.2011.03.007. URL http://www.sciencedirect.com/science/article/pii/S030917081100056X.

for C, X.-R. and C++. XML-RPC for C and C++, 2015. URL http://xmlrpc-c.sourceforge.net/. Accessed: 2015-03-16.

Fortune, D., Gijsbers, P., Gregersen, J., and Moore, R. *OpenMI – Real Progress Towards Integrated Modelling*, volume 68 of *Water Science and Technology Library*. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-79880-4. doi: 10.1007/978-3-540-79881-1_32. URL http://link.springer.com/chapter/10.1007/978-3-540-79881-1_32.

Foundation, O. S. G. Open Source Geospatial Foundation, 2015a. URL http://www.geoserver.org/. Accessed: 2015-03-16.

Foundation, P. S. Python Software Foundation, 2015b. URL http://www.python.org/. Accessed: 2015-03-16.

Foundation, T. A. S. ws-xmlrpc - Apache XML-RPC, Feb 2010. URL http://ws.apache.org/xmlrpc/. Accessed: 2015-03-16.

Fowler, A. A Swing Architecture Overview, Jun 2013. URL http://www.oracle.com/technetwork/java/architecture-142923.html. Accessed: 2015-03-16.

Freeman, E., Freeman, E., Bates, B., and Sierra, K. *Head First Design Patterns*. O'Reilly Media, 1 edition, 2004. ISBN 9780596007126. URL http://oreilly.com/catalog/9780596007126/.

Gamma, E., Helm, R., and Johnson, R. E. *Design Patterns. Elements of Reusable Object-Oriented Software.* Addison-Wesley Longman, Amsterdam, 1st ed., reprint. edition, Oct 1994. ISBN 9780201633610. URL http://www.informit.com/store/design-patterns-elements-of-reusable-object-oriented-9780201633610.

Garcia-Molina, H., Ullman, J., and Widom, J. *Database Systems: The Complete Book*. Pearson Education. Pearson Prentice Hall, 2009. ISBN 9780131354289.

URL http://catalogue.pearsoned.co.uk/catalog/academic/product/0,1144,0131354280-IS,00.html.

GeoExt community. GeoExt, 2015. URL http://geoext.org/. Accessed: 2015-03-16.

Geoportal. Geoportal, 2015. URL http://www.geoportal.de/. Accessed: 2015-03-16.

Geoportal Brandenburg. Geoportal, 2015. URL http://geoportal.brandenburg.de/. Accessed: 2015-03-16.

Gerwin, W., Schaaf, W., Biemelt, D., Fischer, A., Winter, S., and Hüttl, R. F. The artificial catchment "chicken creek" (lusatia, germany)–a landscape laboratory for interdisciplinary studies of initial ecosystem development. *Ecological Engineering*, 35:1786–1796, 2009. doi: 10.1016/j.ecoleng.2009.09.003. URL http://www.sciencedirect.com/science/article/B6VFB-4XFXSS5-3/2/ce2111b6528a7e336988c8d74053099a.

Gibbings, J. *Dimensional Analysis*. Springer, 2011. ISBN 9781849963169.

GNU Project. GCC, the GNU Compiler Collection, 2015. URL https://gcc.gnu.org/. Accessed: 2015-03-16.

Goetz, B. Java theory and practice: Dynamic compilation and performance measurement, Dec 2004. URL http://www.ibm.com/developerworks/java/library/j-jtp12214/. Accessed: 2015-03-16.

Goodall, J. L., Robinson, B. F., and Castronova, A. M. Modeling water resource systems using a service-oriented computing paradigm. *Environmental Modelling & Software*, 26(5):573 – 582, 2011. ISSN 1364-8152. doi: 10.1016/j.envsoft.2010.11.013. URL http://www.sciencedirect.com/science/article/pii/S1364815210003178.

Google. Google Earth, 2015. URL http://www.google.com/earth/. Accessed: 2015-03-16.

Google. Google Maps, 2015. URL http://maps.google.com/. Accessed: 2015-03-16.

Gordon, N., McMahon, T., Finlayson, B., Gippel, C., and Nathan, R. *Stream Hydrology: An Introduction for Ecologists*. Wiley, 2004. ISBN 978-0-470-84358-1. URL http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0470843586.html.

Gottardi, G. and Venutelli, M. A control-volume finite-element model for two-dimensional overland flow. *Advances in Water Resources*, 16(5):277 – 284, 1993. ISSN 0309-1708. doi: 10.1016/0309-1708(93)90019-C. URL http://www.sciencedirect.com/science/article/pii/030917089390019C.

Gregersen, J. B., Gijsbers, P. J. A., and Westen, S. J. P. Openmi: Open modelling interface. *Journal of Hydroinformatics*, 9(3):175–191, 2007. doi: 10.2166/hydro.2007.023. URL http://www.iwaponline.com/jh/009/jh0090175.htm.

Gruber, J. Markdown Syntax Documentation, 2004. URL http://daringfireball.net/projects/markdown/syntax. Accessed: 2015-03-16.

GTK+. The GTK+ Project, 2015. URL http://www.gtk.org/. Accessed: 2015-03-16.

Gunduz, O. and Aral, M. M. River networks and groundwater flow: a simultaneous solution of a coupled system. *Journal of Hydrology*, 301(1–4):216 – 234, 2005. ISSN 0022-1694. doi: 10.1016/j.jhydrol.2004.06.034. URL http://www.sciencedirect.com/science/article/pii/S0022169404003269.

Haverkamp, R., Kutilek, M., Parlange, J., Rendon, L., and Krejca, M. Infiltration under ponded conditions. 2. infiltration equations tested for parameter time-dependence and predictive use. *Soil Science*, 145(5):317–329, May 1988. ISSN 0038-075x.

Hervouet, J.-M. *Hydrodynamics of free surface flows: modelling with the finite element method*. Wiley, 2007. ISBN 0470035587.

Hinkelmann, R. *Efficient Numerical Methods and Information-Processing Techniques for Modeling Hydro- and Environmental Systems*. Springer, 2005. ISBN 3-540-24146-9.

Hinkelmann, R. and Zehe, E. Kopplung von strömungs- und transportprozessen für die modellierung von großhangbewegungen. *Hydrologie und Wasserbewirtschaftung*, 2006.

Hinkelmann, R. and Zehe, E. DFG Forschergruppe 581 „Großhang": Kopplung von Strömungs- und Deformationsprozessen für die Modellierung von Großhangbewegungen, 2013. Abschlussbericht.

IBM. Information Management System, Mar 2012. URL http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/ibmims/. Accessed: 2015-03-16.

ISO. Geographic information – Metadata, 2003. URL http://www.iso.org/iso/catalogue_detail?csnumber=26020. Accessed: 2015-03-16.

ISO. Information processing – Text and office systems – Standard Generalized Markup Language (SGML), 2005. URL http://www.iso.org/iso/catalogue_detail?csnumber=16387. Accessed: 2015-03-16.

ISO. Date and time format - ISO 8601, 2015. URL http://www.iso.org/iso/iso8601. Accessed: 2015-03-16.

JDOM. JDOM, 2015. URL http://jdom.org/. Accessed: 2015-03-16.

Josefsson, S. RFC 4648 - The Base16, Base32, and Base64 Data Encodings, Oct 2006. URL http://tools.ietf.org/html/rfc4648. Accessed: 2015-03-16.

JSON. JSON, 2015. URL http://json.org. Accessed: 2015-03-16.

JSON-RPC Working Group. JSON-RPC 2.0 Specification, Mar 2010. URL http://www.jsonrpc.org/specification. Accessed: 2015-03-16.

JsonRpc-Cpp. JsonRpc-Cpp, 2011. URL http://jsonrpc-cpp.sourceforge.net/. Accessed: 2013-03-27.

Kamfonas, M. J. Recursive Hierarchies: The Relational Taboo!, Oct 1992. URL http://www.kamfonas.com/id3.html. Accessed: 2015-03-16.

Karypis, G. and Kumar, V. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998. doi: 10.1137/S1064827595287997. URL http://dx.doi.org/10.1137/S1064827595287997.

Karypis Lab. Family of Graph and Hypergraph Partitioning Software, 2015. URL http://glaros.dtc.umn.edu/gkhome/views/metis. Accessed: 2015-03-16.

KFKI. NOKIS, 2015. URL http://www.kfki.de/de/service/nokis. Accessed: 2015-03-16.

KISTERS. WISKI Messdatenmanagement, 2015. URL http://www.kisters.de/wasser/software/wiski-messdatenmanagement.html. Accessed: 2015-03-16.

Klensin, J. RFC 5321 - Simple Mail Transfer Protocol, Oct 2008. URL http://tools.ietf.org/html/rfc5321. Accessed: 2015-03-16.

Knapen, R., Janssen, S., Roosenschoon, O., Verweij, P., de Winter, W., Uiterwijk, M., and Wien, J.-E. Evaluating openmi as a model integration platform across disciplines. *Environmental Modelling & Software*, 39(0):274 – 282, 2013. ISSN 1364-8152. doi: 10.1016/j.envsoft.2012.06.011. URL http://www.sciencedirect.com/science/article/pii/S1364815212001946. Thematic Issue on the Future of Integrated Modeling Science and Technology.

Kolditz, O., Delfs, J.-O., Bürger, C., Beinhorn, M., and Park, C.-H. Numerical analysis of coupled hydrosystems based on an object-oriented compartment approach. *Journal of Hydroinformatics*, 10(3):227 – 244, 2008. doi: 10.2166/hydro.2008.003. URL http://dx.doi.org/10.2166/hydro.2008.003.

Kollet, S. J. and Maxwell, R. M. Integrated surface–groundwater flow modeling: A free-surface overland flow boundary condition in a parallel groundwater flow model. *Advances in Water Resources*, 29(7):945 – 958, 2006. doi: 10.1016/j.advwatres.2005.08.006. URL http://www.sciencedirect.com/science/article/pii/S0309170805002101.

Kragt, M. E., Robson, B. J., and Macleod, C. J. Modellers' roles in structuring integrative research projects. *Environmental Modelling & Software*, 39(0):322 – 330, 2013. ISSN 1364-8152. doi: 10.1016/j.envsoft.2012.06.015. URL http://www.sciencedirect.com/science/article/pii/S1364815212001983.

Laurent, S., Johnston, J., and Dumbill, E. *Programming Web Services with XML-RPC*. Creating Web applications gateways. O'Reilly Media, Incorporated, 2001. ISBN 978-0-596-00119-3. URL http://shop.oreilly.com/product/9780596001193.do.

Leach, P. J., Berners-Lee, T., Mogul, J. C., Masinter, L., Fielding, R. T., and Gettys, J. RFC 2616 - Hypertext Transfer Protocol – HTTP/1.1, June 1999. URL http://tools.ietf.org/html/rfc2616. Accessed: 2015-03-16.

Lee Hutchinson. Solid-state revolution: in-depth on how SSDs really work, Jun 2012. URL http://arstechnica.com/information-technology/2012/06/inside-the-ssd-revolution-how-solid-state-disks-really-work/. Accessed: 2015-03-16.

Lindholm, T., Yellin, F., Bracha, G., and Buckley, A. The Java Virtual Machine Specification, 2013. URL http://docs.oracle.com/javase/specs/jvms/se7/html/. Accessed: 2015-03-16.

Mapbender. Mapbender, 2015. URL http://www.mapbender.org/. Accessed: 2015-03-16.

MapWindow GIS. MapWindow Open Source GIS, 2015. URL http://www.mapwindow.org/. Accessed:2015-03-16.

Microsoft. Bing Maps, 2015. URL http://www.bing.com/maps/. Accessed: 2015-03-16.

Microsoft. Interprocess Communications, 2015. URL https://msdn.microsoft.com/en-us/library/windows/desktop/aa365574(v=vs.85).aspx. Accessed: 2015-03-16.

Mirzaee, S., Zolfaghari, A. A., Gorji, M., Dyck, M., and Dashtaki, S. G. Evaluation of infiltration models with different numbers of fitting parameters in different soil texture classes. *Archives of Agronomy and Soil Science*, 60(5):681–693, May 2014. doi: 10.1080/03650340.2013.823477.

Molkenthin, F. *WWW Based Hydroinformatics Systems*. Progress in Bauinformatik. Institut für Bauinformatik, 2000. ISBN 3-934934-04-8.

Molkenthin, F., Notay, V., and Li, C. Hybrid hydroinformatics system for an interdisciplinary research project. In *33rd IAHR congress Vancouver Canada*, 2009a.

Molkenthin, F., Notay, V., Li, C., Hinkelmann, R., and Stadler, L. Model integration and coupling in a hydroinformatics system. volume 01, pages 218–225. Department of Civil Engineering, Technische Universität Berlin, 2009b.

Mono Project. About Mono, 2015. URL http://www.mono-project.com/docs/about-mono/. Accessed: 2015-03-16.

Netlib. BLAS, 1979a. URL http://www.netlib.org/blas/. Accessed: 2015-03-16.

Netlib. LINPACK, 1979b. URL http://www.netlib.org/linpack/. Accessed: 2015-03-16.

Netlib. LAPACK, Nov 2013. URL http://www.netlib.org/lapack/. Accessed: 2015-03-16.

Netlib. LAPACK FAQ, 2015. URL http://www.netlib.org/lapack/faq.html. Accessed: 2015-03-16.

Notay, K. V., Mendoza-Lera, C., Federlein, L. L., and Molkenthin, F. A coupled subsurface-flow and metabolism model to study the effects of solute fluxes in the hyporheic zone. In *EnviroInfo 2014 Conference, Oldenburg, Germany*, 2014. Paper and oral presentation.

NRC. Blue Kenue, 2015. URL http://www.nrc-cnrc.gc.ca/eng/solutions/advisory/blue_kenue_index.html. Accessed: 2015-03-16.

NumPy. NumPy, 2015. URL http://www.numpy.org/. Accessed: 2015-03-16.

Néelz, S. and Pender, G. *Benchmarking of 2D Hydraulic Modelling Packages*. Environmental Agency, 2007. URL http://web.archive.org/web/20120617084637/http://publications.environment-agency.gov.uk/PDF/SCHO0510BSNO-E-E.pdf. Accessed: 2013-03-27.

OASIS. The DocBook Schema Version 5.0, Nov 2009. URL http://docs.oasis-open.org/docbook/specs/docbook-5.0-spec.html. Accessed: 2015-03-16.

OGC. KML, 2008. URL http://www.opengeospatial.org/standards/kml. Accessed: 2015-03-16.

OGC. Simple Feature Access - Part 1: Common Architecture, 2011. URL http://www.opengeospatial.org/standards/sfa. Accessed: 2015-03-16.

OGC. *OGC WaterML 2.0: Part 1 – Timeseries*. OGC, Sep 2012. URL https://portal.opengeospatial.org/files/?artifact_id=48531.

OGC. Geospatial Services, 2015a. URL http://www.ogcnetwork.net/services. Accessed: 2015-03-16.

OGC. Introduction to SensorML, 2015b. URL http://www.ogcnetwork.net/SensorML_Intro. Accessed: 2015-03-16.

OGC. OGC WaterML, 2015c. URL http://www.opengeospatial.org/standards/waterml. Accessed: 2015-03-16.

Open Source Geospatial Foundation. Geonetwork-opensource, 2015. URL http://geonetwork-opensource.org/. Accessed: 2015-03-16.

open TELEMAC-MASCARET. open TELEMAC-MASCARET, 2015. URL http://www.opentelemac.org/. Accessed: 2015-03-16.

OpenFOAM Foundation. The OpenFOAM Foundation, 2015. URL http://www.openfoam.org/. Accessed: 2015-03-16.

OpenLayers. OpenLayers, 2015. URL http://www.openlayers.org/. Accessed: 2015-03-16.

OpenMI. HarmonIT, 2012a. URL https://sites.google.com/a/openmi.org/home/archives/harmonit. Accessed: 2015-03-16.

OpenMI. OpenMI-Life, 2012b. URL https://sites.google.com/a/openmi.org/home/archives/openmi-life. Accessed: 2015-03-16.

OpenMI. New to OpenMI?, 2012c. URL https://sites.google.com/a/openmi.org/home/new-to-openmi. Accessed: 2015-03-16.

OpenMI. OpenMI, 2012d. URL http://www.openmi.org/. Accessed: 2015-03-16.

OpenMI. OpenMI Compliant Components, 2012e. URL https://sites.google.com/a/openmi.org/home/openmi-around-the-world/compliant-components. Accessed: 2013-03-27.

OpenMI. OpenMI Registered Compliant Components, 2012f. URL http://web.archive.org/web/20140823144958/https://sites.google.com/a/openmi.org/home/openmi-around-the-world/compliant-components/registered-compliant-models. Accessed: 2015-03-16.

OpenStreetMap. OpenStreetMap, 2015. URL http://www.openstreetmap.org/. Accessed: 2015-03-16.

Oracle. JavaFX - The Rich Client Platform, Feb 2014. URL http://www.oracle.com/technetwork/java/javase/overview/javafx-overview-2158620.html. Accessed: 2015-03-16.

Oracle. PATH and CLASSPATH, 2015a. URL http://docs.oracle.com/javase/tutorial/essential/environment/paths.html. Accessed: 2015-03-16.

Oracle. Netbeans IDE, 2015b. URL https://netbeans.org. Accessed: 2015-03-16.

OSGeo. GRASS GIS, 2015. URL http://grass.osgeo.org/. Accessed: 2015-03-16.

Österr. Wasser-und Abfallwirtschaftsverband. *Fließgewässermodellierung - Arbeitsbehelf Hydrodynamik*. Bundesministerium für Land- und Forstwirtschaft, Umwelt und Wasserwirtschaft, 2007. URL http://www.lebensministerium.at/publikationen/wasser/flieszgewaesser/flieszgewaessermodellierung-arbeitsbehelf_hydrodynamik.html. Accessed: 2013-03-27.

PostGIS. PostGIS Reference, 2015a. URL http://www.postgis.org/docs/reference.html. Accessed: 2015-03-16.

PostGIS. PostGIS – Spatial and Geographic Objects for PostgreSQL, 2015b. URL http://postgis.net/. Accessed: 2015-03-16.

QGIS Community. QGIS User Guide, 2013. URL http://web.archive.org/web/20130926162440/http://qgis.org/en/docs/user_manual/preamble/features.html. Accessed: 2015-03-16.

QGIS Community. Quantum GIS, 2015. URL http://qgis.org/. Accessed: 2015-03-16.

Quevauviller, P. *Water System Science and Policy Interfacing*, page 357. Royal Society of Chemistry, 2010. ISBN 1847558615.

Redmond, E., Wilson, J., and Carter, J. *Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement*. Oreilly and Associate Series. Pragmatic Bookshelf, 2012. ISBN 9781934356920. URL http://pragprog.com/book/rwdata/seven-databases-in-seven-weeks.

Richards, L. A. Capillary conduction of liquids through porous mediums. *Physics*, 1:318, 1931. doi: 10.1063/1.1745010.

rJava. rJava - Low-level R to Java interface, 2015. URL http://www.rforge.net/rJava/. Accessed: 2015-03-16.

RPy. RPy, 2012. URL http://rpy.sourceforge.net/. Accessed: 2015-03-16.

Sencha. Ext JS 3.4, 2015. URL http://www.sencha.com/products/extjs3/. Accessed: 2015-03-16.

Sheibley, R. W., Jackman, A. P., Duff, J. H., and Triska, F. J. Numerical modeling of coupled nitrification - ìdenitrification in sediment perfusion cores from the hyporheic zone of the shingobee river, mn. 26:977–987, 2003. ISSN 0309-1708. URL http://www.sciencedirect.com/science/article/pii/S0309170803000885.

Silberschatz, A., Korth, H. F., and Sudarshan, S. *Database System Concepts*. McGraw-Hill, 6th edition, Jan 2010. ISBN 0-07-352332-1. URL http://www.db-book.com/.

Simons, F., Busse, T., Hou, J., Özgen, I., and Hinkelmann, R. A model for overland flow and associated processes within the hydroinformatics modelling system. *Journal of Hydroinformatics*, 16(2):375 – 391, 2014. doi: 10.2166/hydro.2013.173.

Singh, V. and Bhallamudi, S. M. Conjunctive surface-subsurface modeling of overland flow. *Advances in Water Resources*, 21(7):567 – 579, 1998. ISSN 0309-1708. doi: 10.1016/S0309-1708(97)00020-1. URL http://www.sciencedirect.com/science/article/pii/S0309170897000201.

Smith, R. E. and Woolhiser, D. A. Overland flow on and infiltrating surface. *Water Resource Research*, 7(4):899–913, 1971. doi: 10.1029/WR007i004p00899.

Spanoudaki, K., Stamou, A. I., and Nanou-Giannarou, A. Development and verification of a 3-d integrated surface water–groundwater model. *Journal of Hydrology*, 375(3–4):410 – 427, 2009. ISSN 0022-1694. doi: 10.1016/j.jhydrol.2009.06.041. URL http://www.sciencedirect.com/science/article/pii/S0022169409003795.

Stadler, L., Hinkelmann, R., and Helmig, R. Modeling macroporous soils with a two-phase dual-permeability model. *Transport in Porous Media*, 95:585–601, 2012.

ISSN 0169-3913. doi: 10.1007/s11242-012-0064-3. URL http://dx.doi.org/10.1007/s11242-012-0064-3.

Stephenson, D. and Meadows, M. *Kinematic hydrology and modelling*. Elsevier, Amsterdam [etc.], 1986.

Succi, S., Sbragaglia, M., and Ubertini, S. Lattice boltzmann method. *Scholarpedia*, 5(5):9507, 2010. doi: 10.4249/scholarpedia.9507. URL http://www.scholarpedia.org/article/Lattice_Boltzmann_Method.

Tatard, L., Planchon, O., Wainwright, J., Nord, G., Favis-Mortlock, D., Silvera, N., Ribolzi, O., Esteves, M., and Huang, C. H. Measurement and modelling of high-resolution flow-velocity data under simulated rainfall on a low-slope sandy soil. *Journal of Hydrology*, 348(1–2):1 – 12, 2008. ISSN 0022-1694. doi: 10.1016/j.jhydrol.2007.07.016. URL http://www.sciencedirect.com/science/article/pii/S0022169407004301.

TELEMAC-MASCARET, O. *TELEMAC-2D User Manual*. EDF, v7p0 edition, Dec 2014. URL http://www.opentelemac.org/downloads/MANUALS/TELEMAC-2D/telemac-2d_user_manual_en_v7p0.pdf.

TeX Users Group. TeX Users Group, 2015. URL http://tug.org/. Accessed: 2015-03-16.

Texas A&M University. SWAT | Soil and Water Assessment Tool, 2015. URL http://swat.tamu.edu/. Accessed: 2015-03-16.

The Eclipse Foundation. Eclipse, 2015. URL http://eclipse.org. Accessed: 2015-03-16.

The Linux Documentation Project. Linux Interprocess Communications, 1996. URL http://www.tldp.org/LDP/lpg/node7.html. Accessed: 2015-03-16.

The matplotlib development team. matplotlib: python plotting, Oct 2015. URL http://matplotlib.org/. Accessed: 2015-03-16.

The OpenMI Association Technical Committee. *OpenMI Document Series: OpenMI Standard 2 Specification for the OpenMI (Version 2.0)*. OpenMI Association, Nov 2010a. URL https://sites.google.com/a/openmi.org/home/learning-more/OpenMIStandard2InterfaceSpecification.pdf.

The OpenMI Association Technical Committee. *What's New in OpenMI 2.0*. OpenMI Association, Oct 2010b. URL https://sites.google.com/a/openmi.org/home/learning-more/WhatsNewinOpenMI2.pdf.

The PostgreSQL Global Development Group. The PostgreSQL Global Development Group, 2015. URL http://www.postgresql.org/. Accessed: 2015-03-16.

The Qt Company. Qt | Cross-platform application and UI development framework, 2015. URL http://www.qt.io/. Accessed: 2015-03-16.

The R Foundation for Statistical Computing. The R Project for Statistical Computing, 2015. URL http://www.r-project.org/. Accessed: 2015-03-16.

The Regional Computing Centre of Erlangen. RRZE Icon set, 2014. URL https://github.com/RRZE-PP/rrze-icon-set. Accessed: 2015-03-16.

Thompson, J., Soni, B., and Weatherill, N. *Handbook of Grid Generation*. Taylor & Francis, 2010. ISBN 9780849326875.

Troch, P., van Loon, E., and Hilberts, A. Analytical solutions to a hillslope-storage kinematic wave equation for subsurface flow. *Advances in Water Resources*, 25(6):637 – 649, 2002. ISSN 0309-1708. doi: 10.1016/S0309-1708(02)00017-9. URL http://www.sciencedirect.com/science/article/pii/S0309170802000179.

UserLand Software. XML-RPC Specification, Jun 2003. URL http://xmlrpc.scripting.com/spec.html. Accessed: 2015-03-16.

USGS. USGS Hydrologic Markup Language, Sep 2012. URL http://water.usgs.gov/XML/NWIS/nwis_hml.htm. Accessed: 2015-03-16.

Vaish, G. *NoSQL Starter*. Packt Publishing, Limited, 2013. ISBN 9781849694988. URL http://www.packtpub.com/nosql-using-scenario-driven-case-studies/book.

Verhoest, N. E. C. and Troch, P. A. Some analytical solutions of the linearized boussinesq equation with recharge for a sloping aquifer. *Water Resources Research*, 36(3):793–800, 2000. ISSN 1944-7973. doi: 10.1029/1999WR900317. URL http://dx.doi.org/10.1029/1999WR900317.

Versteeg, H. and Malalasekera, W. *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*. Pearson Education Limited, 2nd rev. edition, 2007. ISBN 9780131274983. URL http://www.pearson.ch/HigherEducation/PrenticeHall/1471/9780131274983/An-Introduction-to-Computational.aspx.

W3C. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), Apr 2007a. URL http://www.w3.org/TR/soap12-part1/. Accessed: 2015-03-16.

W3C. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, Jun 2007b. URL http://www.w3.org/TR/wsdl20/. Accessed: 2015-03-16.

W3C. Extensible Markup Language (XML) 1.0 (Fifth Edition), Nov 2008. URL http://www.w3.org/TR/REC-xml/. Accessed: 2015-03-16.

W3C. HTML 5: A vocabulary and associated APIs for HTML and XHTML, Oct 2014. URL http://www.w3.org/TR/wsdl20/. Accessed: 2015-03-16.

Web3D Consortium. X3D V3.3 Abstract Specification, Nov 2015. URL http://www.web3d.org/content/x3d-v33-abstract-specification. Accessed: 2015-03-16.

Weisstein, E. W. Polytope, 2003a. URL http://web.archive.org/web/20131016031721/http://mathworld.wolfram.com/Polytope.html. Accessed: 2015-03-16.

Weisstein, E. W. Tensor, 2003b. URL http://mathworld.wolfram.com/Tensor.html. Accessed: 2015-03-16.

Weisstein, E. W. Coordinate System, 2003c. URL http://mathworld.wolfram.com/CoordinateSystem.html. Accessed: 2015-03-16.

Weisstein, E. W. L^2-norm, 2003d. URL http://mathworld.wolfram.com/L2-Norm.html. Accessed: 2015-03-16.

White, F. *Fluid Mechanics*. McGraw-Hill Series in Mechanical Engineering. McGraw-Hill, 2003. ISBN 9780072402179.

Wikipedia. FLOPS, 2015a. URL http://en.wikipedia.org/wiki/Big_data. Accessed: 2015-03-16.

Wikipedia. Inter-process communication, 2015b. URL http://en.wikipedia.org/wiki/Inter-process_communication. Accessed: 2015-03-16.

Wikipedia. JSON-RPC, 2015c. URL http://en.wikipedia.org/wiki/JSON-RPC#Implementations. Accessed: 2015-03-16.

Wikipedia. Julian day, 2015d. URL http://en.wikipedia.org/wiki/Julian_day. Accessed: 2015-03-16.

Wikipedia. List of Java virtual machines, 2015e. URL http://en.wikipedia.org/wiki/List_of_Java_virtual_machines. Accessed: 2015-03-16.

Wikipedia. Markup language, 2015f. URL http://en.wikipedia.org/wiki/Markup_language. Accessed: 2015-03-16.

Wikipedia. Network model, 2015g. URL http://en.wikipedia.org/wiki/Network_model. Accessed: 2015-03-16.

Wikipedia. Comparison of object database management systems, 2015h. URL https://en.wikipedia.org/wiki/Comparison_of_object_database_management_systems. Accessed: 2015-03-16.

Wikipedia. Comparison of object-relational database management systems, 2015i. URL https://en.wikipedia.org/wiki/Comparison_of_object-relational_database_management_systems. Accessed: 2015-03-16.

Wikipedia. Polytope, 2015j. URL http://en.wikipedia.org/wiki/Polytope. Accessed: 2015-03-16.

Wikipedia. PostGIS, 2015k. URL http://en.wikipedia.org/wiki/PostGIS. Accessed: 2015-03-16.

Wikipedia. Comparison of relational database management systems, 2015l. URL https://en.wikipedia.org/wiki/Comparison_of_relational_database_management_systems. Accessed: 2015-03-16.

Wikipedia. Tensor, 2015m. URL http://en.wikipedia.org/wiki/Tensor. Accessed: 2015-03-16.

Wikipedia. Unix time, 2015n. URL http://en.wikipedia.org/wiki/Unix_time. Accessed: 2015-03-16.

Wikipedia. Write once, compile anywhere, 2015o. URL http://en.wikipedia.org/wiki/Write_once,_compile_anywhere. Accessed: 2015-03-16.

Wikipedia. XML-RPC, 2015p. URL http://en.wikipedia.org/wiki/XML-RPC#Implementations. Accessed: 2015-03-16.

WSV. Höhensysteme, 2015a. URL https://www.pegelonline.wsv.de/gast/hilfe#hilfe_hoehensystem. Accessed: 2013-03-27.

WSV. PEGELONLINE, 2015b. URL https://www.pegelonline.wsv.de/. Accessed: 2015-03-16.

Zhu, Y., Shi, L., Lin, L., Yang, J., and Ye, M. A fully coupled numerical modeling for regional unsaturated–saturated water flow. *Journal of Hydrology*, 475(0):188 – 203, 2012. ISSN 0022-1694. doi: 10.1016/j.jhydrol.2012.09.048. URL http://www.sciencedirect.com/science/article/pii/S0022169412008657.